Programmability Configuration Guide, Cisco IOS XE Everest 16.5.1a (Catalyst 3650 Switches)

First Published: 2017-05-31

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

CONTENTS

CHAPTER 1 **New and Changed Information** 1

New and Changed Feature Information 1

PART I **Provisioning** 3

CHAPTER 2 **Zero-Touch Provisioning** 5

Finding Feature Information 5

Information About Zero-Touch Provisioning 5

Zero-Touch Provisioning Overview 5

DHCP Server Configuration for Zero-Touch Provisioning 6

Sample Zero-Touch Provisioning Configurations 6

Sample DHCP Server Configuration on a Management Port 6

Sample DHCP Server Configuration on a Forwarding Port 7

Sample DHCP Server Configuration on a Linux Ubuntu Device 7

Sample Python Script on a TFTP Server 7

Zero-Touch Provisioning Boot Log 8

Additional References for Zero-Touch Provisioning 9

Feature Information for Zero-Touch Provisioning 10

CHAPTER 3 **iPXE** 11

Finding Feature Information 11

Information About iPXE 11

About iPXE 11

iPXE Overview 12

IPv6 iPXE Network Boot 14

IPv6 Address Assignment in ROMMON Mode 16

iPXE-Supported DHCP Options 16
DHCPv6 Unique Identifiers 18
How to Configure iPXE 18
Configuring iPXE 18
Configuring Device Boot 19
Configuration Examples for iPXE 20
Example: iPXE Configuration 20
Sample iPXE Boot Logs 20
Sample DHCPv6 Server Configuration for iPXE 21
Troubleshooting Tips for iPXE 22
Additional References for iPXE 23
Feature Information for iPXE 24

PART II Shells and Scripting 25

CHAPTER 4 Guest Shell 27
Finding Feature Information 27
Information About Guest Shell 27
Guest Shell Overview 27
Guest Shell Vs Guest Shell Lite 28
Guest Shell Security 28
Hardware Requirements for Guestshell 29
Guest Shell Storage Requirements 29
Accessing Guest Shell on a Device 29
Accessing Guest Shell Through the Management Port 30
IOx Overview 30
How to Enable Guest Shell 30
Managing IOx 30
Managing the Guest Shell 32
Enabling and Running the Guest Shell 33
Disabling and Destroying the Guest Shell 34
Accessing the Python Interpreter 34
Configuration Examples for Guest Shell 34
Example: Managing the Guest Shell 34

Sample VirtualPortGroup Configuration 35
Example: Guest Shell Usage 36
Example: Guest Shell Networking Configuration 36
Sample DNS Configuration for Guest Shell 36
Example: Configuring Proxy Environment Variables 36
Example: Configuring Yum and PIP for Proxy Settings 37
Additional References for Guest Shell 37
Feature Information for Guest Shell 38

CHAPTER 5 **Python API** 39
Finding Feature Information 39
Using Python 39
Cisco Python Module 39
Cisco Python Module to Execute IOS CLI Commands 41

CHAPTER 6 **CLI Python Module** 45
Finding Feature Information 45
Information About CLI Python Module 45
About Python 45
Python Scripts Overview 45
Interactive Python Prompt 46
Python Script 46
Supported Python Versions 47
Updating the Cisco CLI Python Module 48
Additional References for the CLI Python Module 48
Feature Information for the CLI Python Module 49

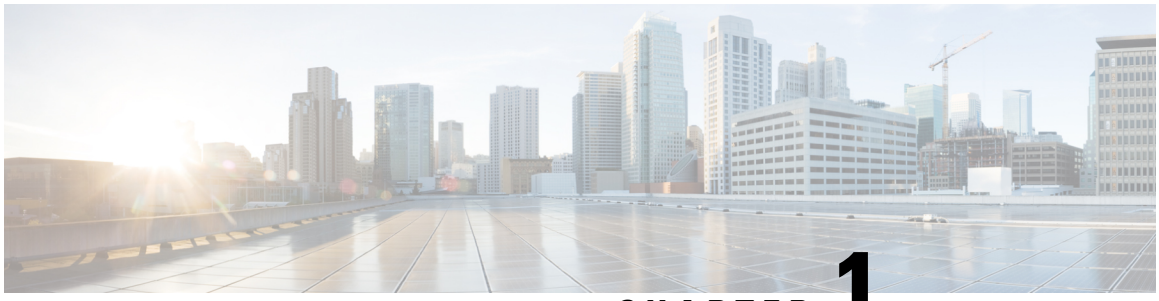
CHAPTER 7 **EEM Python Module** 51
Finding Feature Information 51
Prerequisites for the EEM Python Module 51
Information About the EEM Python Module 51
Python Scripting in EEM 51
EEM Python Package 52
Python-Supported EEM Actions 52

EEM Variables 53
EEM CLI Library Command Extensions 53
How to Configure the EEM Python Policy 54
Registering a Python Policy 54
Running Python Scripts as Part of EEM Applet Actions 56
Adding a Python Script in an EEM Applet 57
Additional References EEM Python Module 59
Feature Information for EEM Python Module 60

PART III **Model-Driven Programmability** 61

CHAPTER 8 **Data Models** 63
Finding Feature Information 63
Restrictions for Data Models 63
Information About Data Models 63
Introduction to Data Models - Programmatic and Standards-Based Configuration 63
NETCONF 64
How to Configure Data Models 64
Configuring NETCONF 64
Configuring NETCONF Options 65
Configuring SNMP 65
Additional References for Data Models 67
Feature Information for Data Models 67

CHAPTER 9 **Operational Data Parser Polling** 69
Finding Feature Information 69
Information About Operational Data 69
Operational Data Overview 69
Operational Data Parsers and Corresponding YANG Models 70
How to Enable Operational Data Parser Polling 70
Enabling Operational Data Parser Polling Through a Programmable Interface 70
Enabling Operational Data Parser Polling Through the CLI 71
Additional References for Operational Data Parser Polling 72
Feature Information for Operational Data Parser Polling 73



CHAPTER 1

New and Changed Information

This chapter provides release-specific information about all features.

- New and Changed Feature Information, on page 1

New and Changed Feature Information

This table summarizes the new and changed features, the supported platforms, and links to features.

Table 1: New and Changed Feature Information

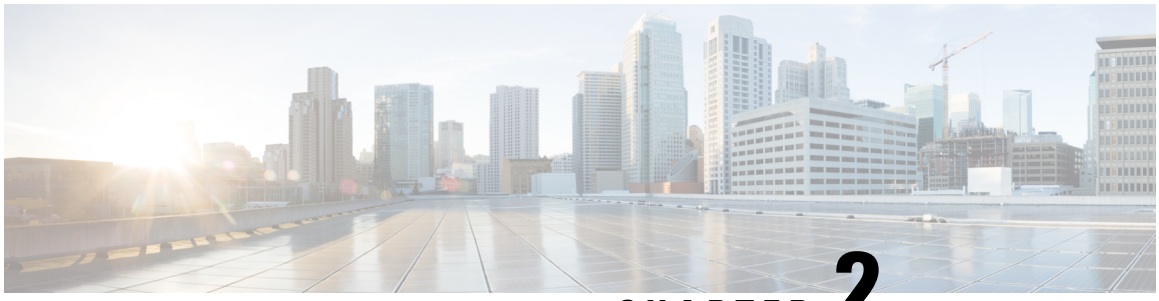
Feature	Description	Release & Platform
Provisioning		
Zero-Touch Provisioning	To address network provisioning challenges, Cisco introduces a Zero-Touch Provisioning model. Zero-Touch Provisioning automates the process of installing or upgrading software images, and installing configuration files on Cisco devices that are deployed in a network for the first time. It reduces manual tasks required to scale the network capacity.	Cisco IOS XE Everest 16.5.1a • Cisco Catalyst 3650 Series Switches • Cisco Catalyst 3850 Series Switches • Cisco Catalyst 9300 Series Switches • Cisco Catalyst 9500 Series Switches
Shells and Scripting		

Feature	Description	Release & Platform
Guest Shell	Guestshell is a virtualized Linux-based environment, designed to run custom Linux applications, including Python for automated control and management of Cisco devices. It also includes the automated provisioning (Day zero) of systems. This container shell provides a secure environment, decoupled from the host device, in which users can install scripts or software packages and run them.	Cisco IOS XE Everest 16.5.1a • Cisco Catalyst 3650 Series Switches • Cisco Catalyst 3850 Series Switches • Cisco Catalyst 9300 Series Switches • Cisco Catalyst 9500 Series Switches
Python APIs	Python programmabililty supports Python APIs.	Cisco IOS XE Everest 16.5.1a • Cisco Catalyst 3650 Series Switches • Cisco Catalyst 3850 Series Switches • Cisco Catalyst 9300 Series Switches • Cisco Catalyst 9500 Series Switches
Python CLI Module	Python Programmability provides a Python module that allows users to interact with IOS using CLIs.	Cisco IOS XE Everest 16.5.1a • Cisco Catalyst 3650 Series Switches • Cisco Catalyst 3850 Series Switches • Cisco Catalyst 9300 Series Switches • Cisco Catalyst 9500 Series Switches
EEM Python Module	Embedded Event Manager (EEM) policies support Python scripts. Python scripts can be executed as part of EEM actions in EEM applets.	Cisco IOS XE Everest 16.5.1a • Cisco Catalyst 3650 Series Switches • Cisco Catalyst 3850 Series Switches • Cisco Catalyst 9300 Series Switches • Cisco Catalyst 9500 Series Switches
Model-Driven Programmability		
Data Models	Cisco IOS XE supports the Yet Another Next Generation (YANG) data modeling language. YANG can be used with the Network Configuration Protocol (NETCONF) to provide the desired solution of automated and programmable network operations.	Cisco IOS XE Denali 16.3.1 • Cisco Catalyst 3650 Series Switches • Cisco Catalyst 3850 Series Switches In Cisco IOS XE Everest 16.5.1a, this feature was implemented on Cisco Catalyst 9300 Series Switches.

PART I

Provisioning

- Zero-Touch Provisioning, on page 5
- iPXE, on page 11



CHAPTER 2

Zero-Touch Provisioning

To address network provisioning challenges, Cisco introduces a zero-touch provisioning model. This module describes the Zero-Touch Provisioning feature.

Note The Zero-Touch Provisioning feature is enabled automatically; no configuration is required.

- Finding Feature Information, on page 5
- Information About Zero-Touch Provisioning, on page 5
- Sample Zero-Touch Provisioning Configurations, on page 6
- Additional References for Zero-Touch Provisioning, on page 9
- Feature Information for Zero-Touch Provisioning, on page 10

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest caveats and feature information, see Bug Search Tool and the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the feature information table at the end of this module.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to https://cfnng.cisco.com/. An account on Cisco.com is not required.

Information About Zero-Touch Provisioning

Zero-Touch Provisioning Overview

To address network provisioning challenges, Cisco introduces a Zero-Touch Provisioning model. Zero-Touch Provisioning automates the process of installing or upgrading software images, and installing configuration files on Cisco devices that are deployed in a network for the first time. It reduces manual tasks required to scale the network capacity.

When a device that supports Zero-Touch Provisioning boots up, and does not find the startup configuration (during fresh install on Day Zero), the device enters the Zero-Touch Provisioning mode. The device locates

a Dynamic Host Control Protocol (DHCP) server, bootstraps itself with its interface IP address, gateway, and Domain Name System (DNS) server IP address, and enables Guest Shell. The device then obtains the IP address or URL of a TFTP server, and downloads the Python script to configure the device.

Guest Shell provides the environment for the Python script to run. Guest Shell executes the downloaded Python script and configures the device for Day Zero.

After Day Zero provisioning is complete, Guest Shell remains enabled. For more information on Guest Shell, see the following chapter:

Note In case Zero-Touch Provisioning fails, the device falls back to AutoInstall to load configuration files. For more information, see Using AutoInstall and Setup.

DHCP Server Configuration for Zero-Touch Provisioning

In Zero-Touch Provisioning, a DHCP server must be running on the same network as the new device that is being provisioned. Zero-Touch Provisioning is supported on both management ports and in-band ports.

When the new device is switched on, it retrieves the IP address information of the TFTP server where the Python script resides, and the folder path of the Python script from the DHCP server.

For more information on Python Scripts, see the following chapters:

The DHCP server responds to DHCP discovery events with the following options:

- Option 150—(Optional) Contains a list of IP addresses that points to the TFTP server on the management network that hosts the Python scripts to be run.
- Option 67—Contains the Python script file path on the TFTP server.

After receiving these DHCP options, the device connects to the TFTP server, and downloads the Python script. The device, at this point does not have any route to reach the TFTP server, so it uses the default route provided by the DHCP server.

Sample Zero-Touch Provisioning Configurations

Sample DHCP Server Configuration on a Management Port

The following is a sample DHCP server configuration when connected via the management port on a device:

```
Device> enable
Device# configure terminal
Device(config)# ip dhcp excluded-address 10.1.1.1
Device(config)# ip dhcp excluded-address vrf Mgmt-vrf 10.1.1.1 10.1.1.10
Device(config)# ip dhcp pool pnp_device_pool
Device(config-dhcp)# vrf Mgmt-vrf
Device(config-dhcp)# network 10.1.1.0 255.255.255.0
Device(config-dhcp)# default-router 10.1.1.1
Device(config-dhcp)# option 150 ip 203.0.113.254
Device(config-dhcp)# option 67 ascii /sample_python_dir/python_script.py
Device(config-dhcp)# no ip dhcp client request tftp-server-address
```

```
Device(config-dhcp)# end
```

Sample DHCP Server Configuration on a Forwarding Port

The following is a sample DHCP server configuration when connected via the forwarding port on a device:

```
Device> enable
Device# configure terminal
Device(config)# ip dhcp excluded-address 10.1.1.1
Device(config)# ip dhcp pool pnp_device_pool
Device(config-dhcp)# network 10.1.1.0 255.255.255.0
Device(config-dhcp)# default-router 10.1.1.1
Device(config-dhcp)# option 150 ip 203.0.113.254
Device(config-dhcp)# option 67 ascii /sample_python_dir/python_script.py
Device(config-dhcp)# no ip dhcp client request tftp-server-address
Device(config-dhcp)# end
```

Sample DHCP Server Configuration on a Linux Ubuntu Device

The following sample DHCP server configuration displays that th server is either connected to the management port or forwarding port on a device. The DHCP server is on a box that is running the Linux Ubuntu distribution.

```
root@ubuntu-server:/etc/dhcp# more dhcpd.conf
subnet 10.1.1.0 netmask 255.255.255.0 {
range 10.1.1.2  10.1.1.255;
      host 3850 {
          fixed-address                       10.1.1.246 ;
          hardware ethernet                   CC:D8:C1:85:6F:00;
          option bootfile-name !<opt 67>      " /python_dir/python_script.py";
         option tftp-server-name !<opt 150>  "203.0.113.254";
      }
  }
```

Once the DHCP server is running, boot a management-network connected device, and the rest of the configuration is automatic.

Sample Python Script on a TFTP Server

The following is a sample Python script hosted on a TFTP server:

```
print "\n\n *** Sample ZTP Day0 Python Script *** \n\n"

# Importing cli module
import cli


print "\n\n *** Executing show platform  *** \n\n"
cli_command = "show platform"
cli.executep(cli_command)

print "\n\n *** Executing show version *** \n\n"
cli_command = "show version"
cli.executep(cli_command)
```

```
print "\n\n *** Configuring a Loopback Interface *** \n\n"
cli.configurep(["interface loop 100", "ip address 10.10.10.10 255.255.255.255", "end"])


print "\n\n *** Executing show ip interface brief  *** \n\n"
cli_command = "sh ip int brief"
cli.executep(cli_command)

print "\n\n *** ZTP Day0 Python Script Execution Complete *** \n\n"
```

Zero-Touch Provisioning Boot Log

The following sample Zero-Touch Provisioning boot log displays that Guest Shell is successfully enabled, the Python script is downloaded to the Guest Shell, and the Guest Shell executes the downloaded Python script and configures the device for Day Zero.

```
% failed to initialize nvram
! <This message indicates that the startup configuration
is absent on the device. This is the first indication that the Day Zero work flow is
going to start.>

This product contains cryptographic features and is subject to United
States and local country laws governing import, export, transfer and
use. Delivery of Cisco cryptographic products does not imply
third-party authority to import, export, distribute or use encryption.
Importers, exporters, distributors and users are responsible for
compliance with U.S. and local country laws. By using this product you
agree to comply with applicable laws and regulations. If you are unable
to comply with U.S. and local laws, return this product immediately.

A summary of U.S. laws governing Cisco cryptographic products may be found at:
http://www.cisco.com/wwl/export/crypto/tool/stqrg.html

If you require further assistance please contact us by sending email to
export@cisco.com.

cisco ISR4451-X/K9 (2RU) processor with 7941237K/6147K bytes of memory.
Processor board ID FJC1950D091
4 Gigabit Ethernet interfaces
32768K bytes of non-volatile configuration memory.
16777216K bytes of physical memory.
7341807K bytes of flash memory at bootflash:.
0K bytes of WebUI ODM Files at webui:.

%INIT: waited 0 seconds for NVRAM to be available


         --- System Configuration Dialog ---

Would you like to enter the initial configuration dialog? [yes/no]: %
!!<DO NOT TOUCH. This is Zero-Touch Provisioning>>
Generating 2048 bit RSA keys, keys will be non-exportable...
[OK] (elapsed time was 1 seconds)
The process for the command is not responding or is otherwise unavailable
The process for the command is not responding or is otherwise unavailable
The process for the command is not responding or is otherwise unavailable
```

```
The process for the command is not responding or is otherwise unavailable
The process for the command is not responding or is otherwise unavailable
The process for the command is not responding or is otherwise unavailable
The process for the command is not responding or is otherwise unavailable
The process for the command is not responding or is otherwise unavailable
The process for the command is not responding or is otherwise unavailable
The process for the command is not responding or is otherwise unavailable
Guestshell enabled successfully


 *** Sample ZTP Day0 Python Script ***


 *** Configuring a Loopback Interface ***


Line 1 SUCCESS: interface loop 100
Line 2 SUCCESS: ip address 10.10.10.10 255.255.255.255
Line 3 SUCCESS: end


 *** Executing show ip interface brief  ***


Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet0/0/0   unassigned      YES unset  down                  down
GigabitEthernet0/0/1   unassigned      YES unset  down                  down
GigabitEthernet0/0/2   unassigned      YES unset  down                  down
GigabitEthernet0/0/3   192.168.1.246   YES DHCP   up                    up
GigabitEthernet0       192.168.1.246   YES DHCP   up                    up
Loopback100            10.10.10.10     YES TFTP   up                    up


 *** ZTP Day0 Python Script Execution Complete ***



Press RETURN to get started!
```

The Day Zero provisioning is complete, and the IOS prompt is accessible.

Additional References for Zero-Touch Provisioning

Related Documents

Related Topic	Document Title
CLI Python Library	
Guest Shell	
iPXE	
Programmability commands	

Technical Assistance

Description	Link
The Cisco Support website provides extensive online resources, including documentation and tools for troubleshooting and resolving technical issues with Cisco products and technologies. To receive security and technical information about your products, you can subscribe to various services, such as the Product Alert Tool (accessed from Field Notices), the Cisco Technical Services Newsletter, and Really Simple Syndication (RSS) Feeds. Access to most tools on the Cisco Support website requires a Cisco.com user ID and password.	http://www.cisco.com/support

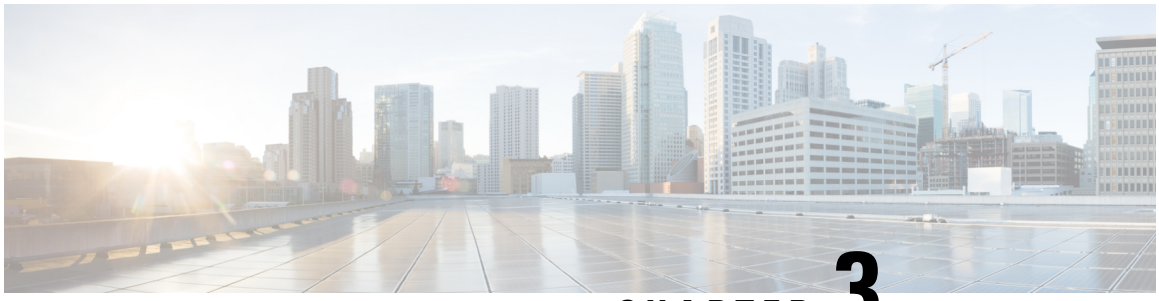
Feature Information for Zero-Touch Provisioning

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 2: Feature Information for Zero-Touch Provisioning

Feature Name	Release	Feature Information
Zero-Touch Provisioning		To address network provisioning challenges, Cisco introduces a zero-touch provisioning model. In Cisco IOS XE Everest 16.5.1a, this feature was implemented on the following platforms:



CHAPTER 3

iPXE

iPXE is an enhanced version of the Pre-boot eXecution Environment (PXE), which is an open standard for network booting. This module describes the iPXE feature and how to configure it.

- Finding Feature Information, on page 11
- Information About iPXE, on page 11
- How to Configure iPXE, on page 18
- Configuration Examples for iPXE, on page 20
- Troubleshooting Tips for iPXE, on page 22
- Additional References for iPXE, on page 23
- Feature Information for iPXE, on page 24

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest caveats and feature information, see Bug Search Tool and the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the feature information table at the end of this module.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to https://cfnng.cisco.com/. An account on Cisco.com is not required.

Information About iPXE

About iPXE

iPXE is an enhanced version of the Pre-boot eXecution Environment (PXE), which is an open standard for network booting.

iPXE netboot provides:

- IPv4 and IPv6 protocols
- FTP/HTTP/TFTP boot image download
- Embedded scripts into the image

- Stateless address auto-configuration (SLAAC) and stateful IP auto-configuration variants for Dynamic Host Configuration Protocol Version 6 (DHCPv6), boot URI, and parameters for DHCPv6 options depending on the IPv6 router advertisement.

Netboot Requirements

The following are the primary requirements for netbooting:

- DHCP server with proper configuration.
- Boot image available on the FTP/HTTP/TFTP server.
- Device configured to boot from a network-based source.

iPXE Overview

Network bootloaders support booting from a network-based source. The bootloaders boot an image located on an HTTP, FTP, or TFTP server. A network boot source is detected automatically by using an iPXE-like solution.

iPXE enables network boot for a device that is offline. The following are the three types of iPXE boot modes:

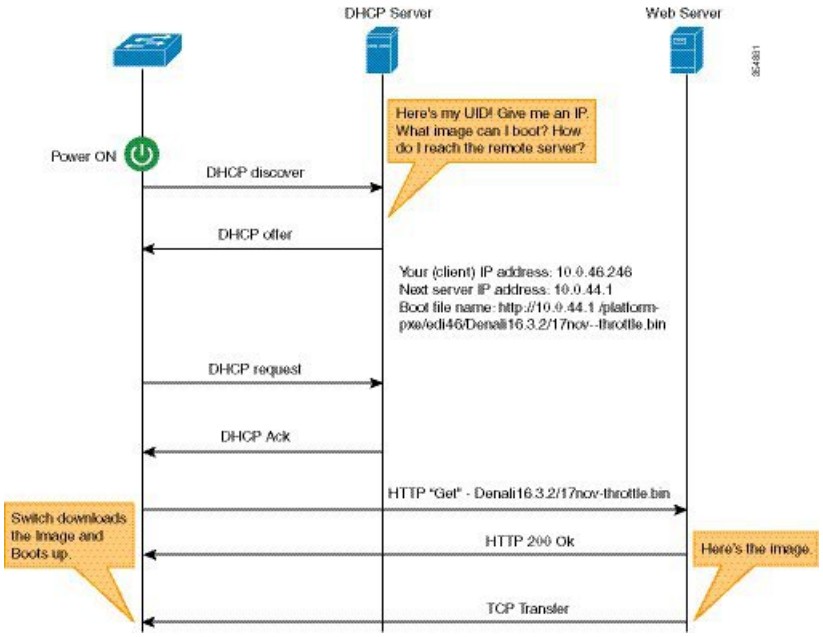
- iPXE Timeout—Configures a timeout in seconds for iPXE network boot by using the IPXE_TIMEOUT rommon variable. When the timeout expires, device boot is activated.
- iPXE Forever—Boots through iPXE network boot. The device sends DHCP requests forever, when the **boot ipxe forever** command is configured. This is an iPXE-only boot (which means that the bootloader will not fall back to a device boot or a command prompt, because it will send DHCP requests forever until it receives a valid DHCP response.)
- Device—Boots using the local device BOOT line configured on it. When device boot is configured, the configured IPXE_TIMEOUT rommon variable is ignored. Device boot is the default boot mode.

Note Manual boot is another term used in this document. Manual boot is a flag that determines whether to do a rommon reload or not. When the device is in rommon mode, you have to manually issue the **boot** command.

If manual boot is set to 1, the rommon or device prompt is activated. If manual boot is set to 0, the device is reloaded; but rommon mode is not activated.

The following section describes how an iPXE bootloader works:

Figure 1: iPXE Bootloader Workflow



1. Bootloader sends a DHCP request.
2. The DHCP response includes the IP address and boot file name. The boot file name indicates that the boot image is to be retrieved from a TFTP server (tftp://server/filename), FTP server (ftp://userid:password@server/filename), or an HTTP server (http://server/filename). Because the current iPXE implementation works only via the management port (GigabitEthernet0/0), DHCP requests sent through the front panel ports are not supported.
3. Bootloader downloads and boots the image from the network source.
4. If no DHCP response is received, the bootloader keeps sending DHCP requests forever or for a specified period of time, based on the boot mode configured. When a timeout occurs, the bootloader reverts to a device-based boot. The device sends DHCP requests forever only if the configured boot mode is **ipxe-forever**. If the **ipxe-timeout** boot mode command is configured, DHCP requests are sent for the specified amount of time, and when the timeout expires, device boot mode is activated.

When manual boot is disabled, the bootloader determines whether to execute a device boot or a network boot based on the configured value of the iPXE ROMMON variable. Irrespective of whether manual boot is enabled or disabled, the bootloader uses the BOOTMODE variable to determine whether to do a device boot or a network boot. Manual boot means that the user has to manually type the **boot manual switch** command to start the boot process. When manual boot is disabled, and when the device reloads, the boot process starts automatically.

When iPXE is disabled, the contents of the existing BOOT variable are used to determine how to boot the device. The BOOT variable may contain a network-based uniform resource identifier (URI) (for example, http://, ftp://, tftp://), and a network boot is initiated; however DHCP is not used to get the network image path. The device IP address is taken from the IP_ADDR variable. The BOOT variable may also contain a device-based path, in which case, a device-based boot is initiated.

To identify the device on a remote DHCP server for booting purposes, use the chassis serial number (available in DHCP option 61), the Product ID (PID) (available in DHCP Option 60), or the device MAC Address. The **show inventory** and **show switch** commands also display these values on the device.

The following is sample output from the show inventory command:

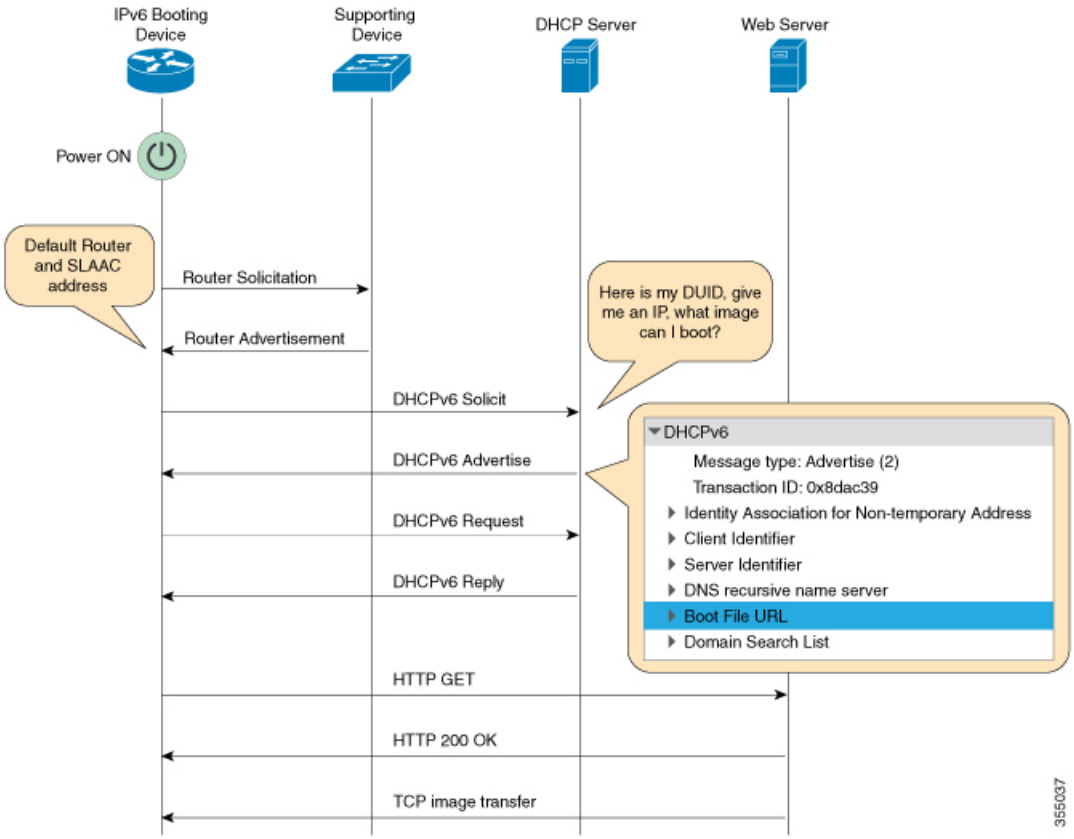
```
Device# show inventory

NAME:"c38xx Stack", DESCR:"c38xx Stack"
PID:WS-3850-12X-48U-L, VID:V01 ,  SN: F0C1911V01A

NAME:"Switch 1", DESCR:"WS-C3850-12X48U-L"
PID:WS-C3850-12X48U-L, VID:V01 , SN:F0C1911V01A

NAME:"Switch1 -Power Supply B", DESCR:"Switch1 -Power Supply B"
PID:PWR-C1-1100WAC, VID:V01, SN:LIT1847146Q
```

The following rommon variables should be configured for iPXE:

- BOOTMODE = ipxe-forever | ipxe-timeout | device
- IPXE_TIMEOUT = seconds

IPv6 iPXE Network Boot

This illustration displays how IPv6 iPXE network boot works on a Cisco device:



The four elements in the above illustration are described below:

- IPv6 Booting Device—The device that is booting through iPXE boot.
- Supporting Device—A Cisco device that is configured with an IPv6 address to generate Router Advertisement (RA) messages.

Note In this illustration, the IPv6 booting device, the supporting device, and the DHCP server are on the same subnet. However; if the supporting device and the DHCP server are on different subnets, then there must be a relay agent in the network.

- DHCP server—Any open source DHCP server.
- Web server—Any open source web server.

This section describes the IPv6 iPXE boot process:

1. The device sends a router solicitation Internet Control Message Protocol IPv6 (ICMPv6) type 133 packet to the IPv6 device on the local subnet.
2. The IPv6 device on the local subnet replies with an RA, ICMPv6 type 134 packet. The device that sent the router solicitation message, gets the default router and prefix information for Stateless Address AutoConfiguration (SLAAC) address completion from the RA packet.
3. The device sends a DHCPv6 solicit message to the multicast group address of ff02::1:2 for all DHCP agents.

   The following sample displays the fields in a DHCPv6 solicit packet during iPXE boot:

```
DHCPv6
 Message type: Solicit (1)
 Transaction ID: 0x36f5f1
 Client Identifier
 Vendor Class
 Identity Association for Non-Temporary Address
 Option Request
 User Class
 Vendor-specific Information
```

   The DHCPv6 solicit message contains the following information:

   - DHCP Unique Identifier (DUID)—Identifies the client. iPXE supports DUID-EN. EN stands for Enterprise Number, and this DUID is based on the vendor-assigned unique identifier.
   - DHCPv6 Option 3
   - DHCPv6 Option 6
   - DHCPv6 Option 15
   - DHCPv6 Option 16
   - DHCPv6 Option 17

4. If the DHCPv6 server is configured, it responds with a DHCPv6 advertise packet that contains the 128 Bit IPv6 address, the boot file Uniform Resource Identifier (URI), the Domain Name System (DNS) server

and domain search list, and the client and server IDs. The client ID contains the DUID of the client (In this illustration, the IPv6 Booting Device), and the Server ID contains the DUID of the DHCPv6 server.

5. The client then sends a DHCPv6 request packet to the multicast group address ff02::1:2, requesting for advertised parameters.
6. The server responds with a unicast DHCPv6 reply to the Link Local (FE80::) IPv6 address of the client. The following sample displays the fields in a DHCPv6 reply packet:

```
DHCPv6
 Message type: Reply (7)
 Transaction ID: 0x790950
 Identity Association for Non-Temporary Address
 Client Identifier
 Server Identifier
 DNS recursive name server
 Boot File URL
 Domain Search List
```

7. The device then sends an HTTP GET request to the web server.
8. If the requested image is available at the specified path, the web server responds with an OK for the HTTP GET request.
9. The TCP image transfer copies the image, and the device boots up.

IPv6 Address Assignment in ROMMON Mode

The DHCP client uses the following order-of-precedence to decide which IPv6 address to use in rommon mode:

1. DHCP Server-assigned address
2. Stateless Address Auto-Configuration (SLAAC) address
3. Link-local address
4. Site-local address

The device uses the DHCP server-assigned address to boot an image. If the DHCPv6 server fails to assign an address, the device tries to use the SLAAC address. If both the DHCP server-assigned address and the SLAAC address are not available, the device uses the link-local address. However, the remote FTP/HTTP/TFTP servers must be on the same local subnet as that of the device for the image copy to succeed.

If the first three addresses are not available, the device uses the automatically generated site-local address.

iPXE-Supported DHCP Options

iPXE boot supports the following DHCPv4 and DHCPv6 options in rommon mode.

- DHCP Option 77—User Class Option. This option is added to a DHCP Discover packet, and contains the value equal to the string *iPXE*. This option helps to isolate iPXE DHCP clients looking for an image to boot from a DHCP server.

The following is sample DHCPv4 configuration from the ISC DHCP Server that displays the use of Option 77. The *if* condition in this sample implies that if Option 77 exists, and is equal to the string *iPXE*, then advertise the Boot File URI for the image.

```
host Switch2 {
     fixed-address 192.168.1.20 ;
     hardware ethernet CC:D8:C1:85:6F:11 ;
           #user-class = length of string + ASCII code for iPXE
     if exists user-class and option user-class = 04:68:50:58:45 {
         filename "http://192.168.1.146/test-image.bin"
     }
}
```

- DHCPv6 Option 15—User Class Option. This option is the IPv6 User Class option in a DHCPv6 solicit message. The following sample shows Option 15 defined in the ISC DHCP server:

```
option dhcp6.user-class code 15 = string ;
```

The following is a sample DHCP Server configuration that uses the DHCPv6 Option 15:

```
#Client-specific parameters
host switch1 {
     #assigning a fixed IPv6 address
     fixed-address6 2001:DB8::CAFE ;
     #Client DUID in hexadecimal format contains: DUID-type"2" + "EN=9" + "Chassis
serial number"
     host-identifier option dhcp6.client-id       00:02:00:00:00:09:46:4F:43:31:38:33:
31:58:31:41:53;
    #User class 00:04:69:50:58:45 is len 4 + "iPXE"
    if option dhcp6.user-class = 00:04:69:50:58:45 {
       option dhcp6.bootfile-url
"http://[2001:DB8::461/platform-pxe/edi46/test-image.bin";
    }
}
```

- DHCPv6 Option 16—Vendor Class Option. Contains the device product ID (PID). The PID can be determined from the output of the **show inventory** command or from the MODEL_NUM rommon variable. Option 16 is not a default option in the ISC DHCP Server and can be defined as follows:

```
option dhcp6.vendor-class-data code 16 = string;
```

The following sample configuration illustrates the use of DHCPv6 Option 16:

```
# Source: dhcpd6ConfigPD

host host1-ipxe6-auto-host1 {
     fixed-address6 2001:DB8::1234;
     host-identifier option dhcp6.client-id 00:02:00:00:00:09:46:4F:
     43:31:38:33:31:58:31:41:53;
     if option dhcp6.vendor-class-data = 00:00:00:09:00:0E:57:53:2D:
     43:33:38:35:30:2D:32:34:50:2D:4D {
     option dhcp6.bootfile-url
"http://[2001:DB8::46]/platform-pxe/host1/17jan-polaris.bin";
```

The table below describes the significant fields shown in the display.

Table 3: Sample Output Field Descriptions

Field	Description
dhcp6.client-id	DHCP Unique Identifier (DUID) to identify the client.
dhcp6.user-class	DHCPv6 Option 15, the User Class option
dhcp6.vendor-class-data	DHCPv6 Option 16, the Vendor Class option that contains the switch Product ID (PID).
N/A	DHCPv6 Option 3 to request for a non-temporary address.
N/A	DHCPv6 Option 17, the Vendor-Specific option that contains the reserved Enterprise ID *9* for Cisco Systems.
dhcp6.bootfile-url	DHCPv6 Option 6 to request for the Boot File URI

DHCPv6 Unique Identifiers

There are three types of DHCPv6 Identifiers (DUIDs) defined by RFC 3315; these are:

- DUID-LLT—DUID Link Layer address plus time, this is the link layer address of the network interface connected to the DHCP device plus the time stamp at which it is generated.
- DUID-EN—EN stands for Enterprise Number, this DUID is based on vendor-assigned unique ID.
- DUID-LL—DUID formed using the Link Layer address of any network interface that is permanently connected to the DHCP (client/server) device.

Cisco devices use the DUID-EN (DUID Type 2) to identify the DHCP client (that is the device in the DHCPv6 Solicit packet).

How to Configure iPXE

Configuring iPXE

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.

	Command or Action	Purpose
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.

Step 3	• **boot ipxe forever** *switch number*
	• **boot ipxe timeout** *seconds switch number*
	Example:
	`Device(config)# boot ipxe forever switch 2`
	Example:
	`Device(config)# boot ipxe timeout 30 switch 2`
		Configures the BOOTMODE rommon variable.
		• The **forever** keyword configures the BOOTMODE rommon variable as IPXE-FOREVER.
		• The **timeout** keyword configures the BOOTMODE rommon variable as IPXE-TIMEOUT.

Step 4	**boot system** {**switch** *switch-number* | **all**} {**flash:** | **ftp:** | **http:** | **tftp:**}
	Example:
	`Device(config)# boot system switch 1 http://192.0.2.42/image-filename`
	or
	`Device(config)# boot system switch 1 http://[2001:db8::1]/image-filename`
		Boots an image from the specified location.
		• You can either use an IPv4 or an IPv6 address for the remote FTP/HTTP/TFTP servers.
		• You must enter the IPv6 address inside the square brackets (as per RFC 2732); if not the device will not boot.

Step 5	**end** **Example:** `Device(config)# end`	Exits global configuration mode and returns to privileged EXEC mode.

Configuring Device Boot

You can either use the **no boot ipxe** or the **default boot ipxe** command to configure device boot.

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	• **no boot ipxe** • **default boot ipxe** **Example:**	Configures device boot. The default bot mode is device boot. Enables default configuration on the device.

	Command or Action	Purpose

	`Device(config)# no boot ipxe`
	Example:
	`Device(config)# default boot ipxe`	

Step 4	**end** **Example:** `Device(config)# end`	Exits global configuration mode and returns to privileged EXEC mode.

Configuration Examples for iPXE

Example: iPXE Configuration

The following example shows that iPXE is configured to send DHCP requests forever until the device boots with an image:

```
Device# configure terminal
Device(config)# boot ipxe forever switch 2
Device(config)# end
```

The following example shows how to configure the boot mode to ipxe-timeout. The configured timeout is 200 seconds. If an iPXE boot failure occurs after the configured timeout expires, the configured device boot is activated. In this example, the configured device boot is http://[2001:db8::1]/image-filename.

```
Device# configure terminal
Device(config)# boot ipxe timeout 200 switch 2
Device(config)# boot system http://[2001:db8::1]/image-filename
Device(config)# end
```

Sample iPXE Boot Logs

The following are sample boot logs from a device in rommon mode. Here, manual boot using the **ipxe-timeout** command is configured:

```
switch: boot

pxemode:(ipxe-timeout) 60s timeout
00267.887 ipxe_get_booturl: Get URL from DHCP; timeout 60s
00267.953 ipxe_get_booturl: trying DHCPv6 (#1) for 10s
IPv4:
           ip addr 192.168.1.246
           netmask 255.255.255.0
           gateway 192.168.1.46
IPv6:
link-local addr fe80::ced8:c1ff:fe85:6f00
site-local addr fec0::ced8:c1ff:fe85:6f00
```

```
      DHCP addr 2001:db8::cafe
    router addr fe80::f29e:63ff:fe42:4756
     SLAAC addr 2001:db8::ced8:c1ff:fe85:6f00 /64
Common:
        macaddr cc:d8:c1:85:6f:00
            dns 2001:db8::46
       bootfile
http://[2001:DB8::461/platform-pxe/edi46/17jan-dev.bin--13103--2017-Feb28--13-54-50
         domain cisco.com
00269.321 ipxe_get_booturl: got URL
(http://[2001:DB8::461/platform-pxe/edi46/17jan-dev.bin--13103--2017-Feb-28--13-54-50)
Reading full image into memory ….…….…….…….…….…….…….…….…….…….…….…….…...
Bundle Image
---------------------------------------------
Kernel Address    : 0x5377a7e4
Kernel Size       : 0x365e3c/3563068
Initramfs Address : 0x53ae0620
Initramfs Size    : 0x13a76f0/20608752
Compression Format: mzip
```

Sample DHCPv6 Server Configuration for iPXE

The following is a sample DHCPv6 server configuration taken from an ISC DHCP Server for reference. The lines preceded by the character #, are comments that explain the configuration that follows.

```
Default-least-time 600;
max-lease-time-7200;
log-facility local7;

#Global configuration
#domain search list
option dhcp6.domain-search "cisco.com" ;
#User-defined options:new-name code new-code = definition ;
option dhcp6.user-class code 15 = string ;
option dhcp6.vendor-class-data code 16 = string;

subnet6 2001:db8::/64 {
 #subnet range for clients requiring an address
 range6 2001:db8:0000:0000::/64;

#DNS server options
option dhcp6.name-servers 2001:db8::46;

}
#Client-specific parameters
host switch1 {
    #assigning a fixed IPv6 address
    fixed-address6 2001:DB8::CAFE ;
    #Client DUID in hexadecimal that contains: DUID-type "2" + "EN=9" + "Chassis serial
number"
    host-identifier option dhcp6.client-id 00:02:00:00:00:09:46:4F:43:31:38:33:
31:58:31:41:53;
   option dhcp6.bootfile-url "http://[2001:DB8::461/platform-pxe/edi46/test-image.bin";
}
```

For more information on DHCP server commands, see the *ISC DHCP Server* website.

In this sample configuration, the dhcp6.client-id option identifies the switch, and it is followed by the Enterprise Client DUID. The client DUID can be broken down for understanding as 00:02 + 00:00:00:09 + chassis serial number in hexadecimal format, where 2 refers to the Enterprise Client DUID Type, 9 refers to the reserved code for Cisco's Enterprise DUID, followed by the ASCII code for the Chassis serial number in hexadecimal format. The chassis serial number for the switch in this sample is FOC1831X1AS.

The Boot File URI is advertised to the switch only using the specified DUID.

The DHCPv6 Vendor Class Option 16 can also be used to identify the switch on the DHCP Server. By default, this DHCP option is not supported by the ISC DHCP Server, and to define it as a user-defined option, configure the following:

```
option dhcp6.vendor-class-data code 16 = string;
```

The following is a sample DHCP server configuration that identifies the switch based on the DHCPv6 Vendor Class Option 16 that is formed by using the switch Product ID:

```
# Source: dhcp6ConfigPID

host edi-46-ipxe6-auto-edi46 {
    fixed-address6 2001:DB8::1234;
    host-identifier option dhcp6.client-id 00:02:00:00:00:09:
    46:4F:43:31:38:33:31:58:31:58:31:41:53;
    if option dhcp6.vendor-class-data = 00:00:00:09:00:0E:57:
    53:2D:43:33:38:35:30:2D:32:34:50:2D:4C {
       option dhcp6.bootfile-url "http://[2001:DB8::461/platform-pxe/edi46/17jan-dev.bin";
    }
}
```

In this sample configuration, the dhcp6.vendor-class-data option refers to the DHCPv6 Option 16. In the dhcp6.vendor-class-data, 00:00:00:09 is Cisco's Enterprise DUID, 0E is the length of the PID, and the rest is the PID in hexadecimal format. The PID can also be found from the output of the **show inventory** command or from the CFG_MODEL_NUM rommon variable. The PID used in this sample configuration is WS-C3850-24P-L.

DHCPv6 options and DUIDs in the server configuration must be specified in the hexadecimal format, as per the ISC DHCP server guidelines.

Troubleshooting Tips for iPXE

This section provides troubleshooting tips.

- When iPXE boot is enabled on power up, the device first attempts to send a DHCPv6 Solicit message, followed by a DHCPv4 Discover message. If boot mode is **ipxe-forever** the device keeps iterating between the two forever.
- If the boot-mode is iPXE timeout, the device first sends a DHCPv6 Solicit message, and then a DHCPv4 Discover message, and the device falls back to device boot after the timeout expires.
- To interrupt iPXE boot, send a serial break to the console.

  When using a UNIX telnet client, type CTRL-] and then send break. When you are using a different TELNET client, or you are directly attached to a serial port, sending a break may be triggered by a different keystroke or command.

- If the DHCP server responds with an image, but the DNS server cannot resolve the hostname, enable DNS debugs.
- To test the HTTP server connectivity, use HTTP copy to copy a small sample file from your HTTP server to your device. For example, at the rommon prompt, enter **copy http://192.168.1.1/test null:** (the flash is normally locked and you need to use the null device for testing) or **http://[2001:db8::99]/test**.
- When manual boot is enabled, and boot mode is ipxe-timeout, the device will not automatically boot on power up. Issue the **boot** command in rommon mode. To automate the boot process on power up, disable manual boot.
- Use the **net6-show** command to display the current IPv6 parameters, including IPv6 addresses and the default router in rommon mode
- Use the **net-dhcp** or the **net6-dhcp** commands based on your configuration, The **net-dhcp** command is a test command for DHCPv4 and the **net6-dhcp** command is for DHCPv6.
- Use the **dig** command to resolve names.
- Enable HTTP debug logs to view the HTTP response code from the web server.
- If SLAAC addresses are not generated, there is no router that is providing IPv6 RA messages. iPXE boot for IPv6 can still work but only with link or site-local addresses.

For more information about iPXE commands, see the

- Catalust 3650 Command Reference
- Catalust 3850 Command Reference

Additional References for iPXE

Related Documents

Related Topic	Document Title
Programmability commands	

Standards and RFCs

Standard/RFC	Title
RFC 3315	*Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*
RFC 3986	*Uniform Resource Identifier (URI): Generic Syntax*

Technical Assistance

Description	Link
The Cisco Support website provides extensive online resources, including documentation and tools for troubleshooting and resolving technical issues with Cisco products and technologies. To receive security and technical information about your products, you can subscribe to various services, such as the Product Alert Tool (accessed from Field Notices), the Cisco Technical Services Newsletter, and Really Simple Syndication (RSS) Feeds. Access to most tools on the Cisco Support website requires a Cisco.com user ID and password.	http://www.cisco.com/support

Feature Information for iPXE

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

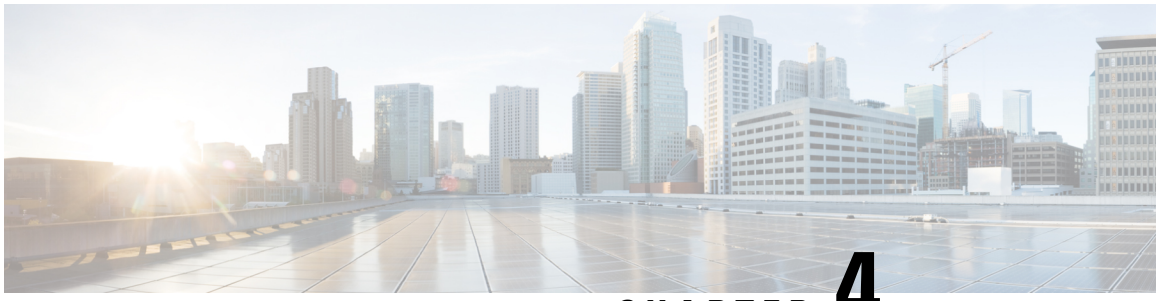
Table 4: Feature Information for iPXE

Feature Name	Release	Feature Information
iPXE		Network Bootloaders support booting from a device-based or network-based source. A network boot source must be detected automatically by using an iPXE-like solution.

PART II

Shells and Scripting

- Guest Shell, on page 27
- Python API, on page 39
- CLI Python Module, on page 45
- EEM Python Module, on page 51



CHAPTER 4

Guest Shell

Guestshell is a virtualized Linux-based environment, designed to run custom Linux applications, including Python for automated control and management of Cisco devices. It also includes the automated provisioning (Day zero) of systems. This container shell provides a secure environment, decoupled from the host device, in which users can install scripts or software packages and run them.

This module describes Guest Shell and how to enable it.

- Finding Feature Information, on page 27
- Information About Guest Shell, on page 27
- How to Enable Guest Shell, on page 30
- Configuration Examples for Guest Shell, on page 34
- Additional References for Guest Shell, on page 37
- Feature Information for Guest Shell, on page 38

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest caveats and feature information, see Bug Search Tool and the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the feature information table at the end of this module.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to https://cfnng.cisco.com/. An account on Cisco.com is not required.

Information About Guest Shell

Guest Shell Overview

Guestshell is a virtualized Linux-based environment, designed to run custom Linux applications, including Python for automated control and management of Cisco devices. Using Guest Shell, customers can also install, update, and operate third-party Linux applications. It is bundled with the system image and can be installed using the **guestshell enable** IOS command.

The Guest Shell environment is intended for tools, Linux utilities, and manageability rather than networking.

Guest Shell shares the kernel with the host (Cisco switches and routers) system. Users can access the Linux shell of Guest Shell and update scripts and software packages in the container rootfs. However, users within the Guest Shell cannot modify the host file system and processes.

Guest Shell container is managed using IOx. IOx is Cisco's Application Hosting Infrastructure for Cisco IOS XE devices. IOx enables hosting of applications and services developed by Cisco, partners, and third-party developers in network edge devices, seamlessly across diverse and disparate hardware platforms.

This table provides information about the various Guest Shell capabilities and the supported platforms.

Table 5: Cisco Guest Shell Capabilities

	Guest Shell Lite (Limited LXC Container)	**Guest Shell (LXC Container)**
Operating System	Cisco IOS XE	Cisco IOS XE
Supported Platforms		
Guest Shell Environment	Montavista CGE7	CentOS 7
Python 2.7	Supported (Python V2.7.11)	Supported (Python V2.7.5)
Custom Python Libraries	• Cisco Embedded Event Manager • Cisco IOS XE CLIs • Ncclient	• Cisco Embedded Event Manager • Cisco IOS XE CLIs
Supported Rootfs	Busybox, SSH, and Python PIP install	SSH, Yum install, and Python PIP install
GNU C Compiler	Not supported	Not supported
RPM Install	Not supported	Supported
Architecture	MIPS	x86

Guest Shell Vs Guest Shell Lite

The Guest Shell container allows users to run their scripts and apps on the system. The Guest Shell container on Intel x86 platforms will be a Linux container (LXC) with a CentOS 7.0 minimal rootfs. You can install other Python libraries such as, Python Version 3.0 during runtime using the Yum utility in CentOS 7.0. You can also install or update python packages using PIP.

The Guest Shell Lite container on MIPS platforms such as, Catalyst 3650 and Catalyst 3850 Series Switches have the Montavista Carrier Grade Edition (CGE) 7.0 rootfs. You can only install or run scripts in Guest Shell Lite. Yum install is not supported on these devices.

Guest Shell Security

Cisco provides security to ensure that users or apps in the Guest Shell do not compromise the host system. Guest Shell is isolated from the host kernel, and it runs as an unprivileged container.

Hardware Requirements for Guestshell

This section provides information about the hardware requirements for supported platforms.

Table 6: Guest Shell Support on Catalyst Switches

Note Virtual-service installed applications and Guest Shell container cannot co-exist.

Guest Shell Storage Requirements

On Catalyst 3650 and Catalyst 3850 Series Switches, Guest Shell can only be installed on the flash filesystem. Bootflash of Catalyst 3850 Series Switches require 75 MB free disk space for Guest Shell to install successfully.

On Cisco 4000 Series Integrated Services Routers, Guest Shell is installed on the Network Interface Module (NIM)-Service Set Identifier (SSD) (hard disk), if available. If the hard disk drive is available, there is no option to select bootflash to install Guest Shell. Cisco 4000 Series Integrated Services Routers require 1100 MB free hard disk (NIM-SSID) space for Guest Shell to install successfully.

During Guest Shell installation, if enough hard disk space is not available, an error message is displayed.

Bootflash or hard disk space can be used to store additional data by Guest Shell. On Cisco Catalyst 3850 Series Switches, Guest Shell has 18 MB of storage space available and on Cisco 4000 Series Integrated Services Routers, Guest Shell has 800 MB of storage space available. Because Guest Shell accesses the bootflash, it can use the entire space available.

Table 7: Resources Available to Guest Shell and Guest Shell Lite

Resource	Default	Minimum/Maximum
CPU	1% **Note** 1% is not standard; 800 CPU units/ total system CPU units.	1/100%
Memory	256 MB	256/256 MB

Accessing Guest Shell on a Device

Network administrators can use IOS commands to manage files and utilities in the Guest Shell.

During the Guest Shell installation, SSH access is setup with a key-based authentication. The access to the Guest Shell is restricted to the user with the highest privilege (15) in IOS. This user is granted access into the Linux container as the *guestshell* Linux user, who is a sudoer, and can perform all root operations. Commands executed through the Guest Shell are executed with the same privilege that a user has when logged into the IOS terminal.

At the Guest Shell prompt, you can execute standard Linux commands.

Accessing Guest Shell Through the Management Port

By default, Guest Shell allows applications to access the management network. Users cannot change the management VRF networking configurations from inside the Guest Shell.

Note For platforms without a management port, a VirtualPortGroup can be associated with Guest Shell in the IOS configuration. For more information, see the *Sample VirtualPortGroup Configuration* section.

IOx Overview

IOx is a Cisco-developed end-to-end application framework that provides application hosting capabilities for different application types on Cisco network platforms. The Cisco Guest Shell, a special container deployment, is one such application, that is useful in system deployment/use.

IOx facilitates the life-cycle management of app and data exchange by providing a set of services that helps developers to package pre-built apps, and host them on a target device. IOx life-cycle management includes distribution, deployment, hosting, starting, stopping (management), and monitoring of apps and data. IOx services also include app distribution and management tools that help users discover and deploy apps to the IOx framework.

App hosting provides the following features:

- Hides network heterogeneity.
- IOx application programming interfaces (APIs), remotely manage the life cycle of applications hosted on a device.
- Centralized app life-cycle management.
- Cloud-based developer experience.

How to Enable Guest Shell

Managing IOx

Before you begin

IOx takes upto two minutes to start. CAF, IOXman, and Libirtd services must be running to enable Guest Shell successfully.

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.

	Command or Action	Purpose
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**iox** **Example:** `Device(config)# iox`	Configures IOx services.
Step 4	**exit** **Example:** `Device(config)# exit`	Exits global configuration mode and returns to privileged EXEC mode.
Step 5	**show iox-service** **Example:** `Device# show iox-service`	Displays the status of the IOx service
Step 6	**show app-hosting list** **Example:** `Device# show app-hosting list`	Displays the list of app-hosting services enabled on the device.

What to do next

The following is sample output from the **show iox-service** command on an ISR 4000 Series Router:

```
Device# show iox-service

Virtual Service Global State and Virtualization Limits:

Infrastructure version : 1.7
Total virtual services installed : 0
Total virtual services activated : 0

Machine types supported   : KVM, LXC
Machine types disabled    : none

Maximum VCPUs per virtual service : 6
Resource virtualization limits:
Name                         Quota     Committed     Available
------------------------------------------------------------------
system CPU (%)                  75             0            75
memory (MB)                  10240             0         10240
bootflash (MB)                1000             0          1000
harddisk (MB)                20000             0         18109
volume-group (MB)           190768             0        170288


IOx Infrastructure Summary:
---------------------------
IOx service (CAF)    : Running
IOx service (HA)     : Not Running
IOx service (IOxman) : Running
Libvirtd             : Running
```

The following is truncated sample output from the **show iox-service** command on a Catalyst 3850 Series Switch:

```
Device# show iox-service

IOx Infrastructure Summary:
---------------------------
IOx service (CAF)    : Running
IOx service (HA)     : Running
IOx service (IOxman) : Running
Libvirtd             : Running
```

The following is sample output from the **show app-hosting list** command:

```
Device# show app-hosting list

App id                           State
---------------------------------------------------------
guestshell                       RUNNING
```

Managing the Guest Shell

You can start the Guest Shell container in IOS through Guest Shell commands.

Before you begin

IOx must be configured and running for Guest Shell access to work. If IOx is not configured, a message to configure IOx is displayed. Removing IOx removes access to the Guest Shell, but the rootfs remains unaffected.

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	• **guestshell enable** • **guestshell enable** [**VirtualPortGroup** *port-number* **guest-ip** *ip-address* **gateway** *gateway-ip* **netmask** *netmask* [**name-server** *ip-address*]]	Enables the Guest Shell service. or Enables connectivity to the front panel ports.

	Command or Action	Purpose

	Example:
	`Device# guestshell enable`
	Example:
	`Device# guestshell enable VirtualPortGroup 0 guest-ip 192.168.35.2 gateway 192.168.35.1 netmask 255.255.255.0 name-server 10.1.1.1`
		Note
		• The **guestshell enable** command without any arguments uses the management virtual routing and forwarding (VRF) instance for networking.
		• When using VirtualPortGroups (VPGs) for front panel networking, the VPG must be configured first.
		• The guest IP address and the gateway IP address must be in the same subnet.
		• Front panel ports are not supported Cisco Catalyst 3650 Series Switches, Cisco Catalyst 3850 Series Switches, Cisco Catalyst 9300 Series Switches, and Cisco Catalyst 9500 Series Switches.

Step 3	**guestshell run** *linux-executable* **Example:** `Device# guestshell run python`	Executes or runs a Linux program in the Guest Shell. • Python Version 2.7.11 is pre-installed on Catalyst 3650 and Catalyst 3850 Series Switches, and Python Version 2.7.5 is pre-installed on ISR 4000 Series Routers.
Step 4	**guestshell run bash** **Example:** `Device# guestshell run bash`	Starts a Bash shell to access the Guest Shell.
Step 5	**guestshell disable** **Example:** `Device# guestshell disable`	Disables the Guest Shell service.
Step 6	**guestshell destroy** **Example:** `Device# guestshell destroy`	Deactivates and uninstalls the Guest Shell service.

Enabling and Running the Guest Shell

The **guestshell enable** command installs Guest Shell. This command is also used to reactivate Guest Shell, if it is disabled.

When Guest Shell is enabled and the system is reloaded, Guest Shell remains enabled.

Note IOx must be configured before the **guestshell enable** command is used.

The **guestshell run bash** command opens the Guest Shell bash prompt. Guest Shell must already be enabled for this command to work.

Note If the following message is displayed on the console, it means that IOx is not enabled; check the output of the **show iox-service** command to view the status of IOx.

```
The process for the command is not responding or is otherwise unavailable
```

Disabling and Destroying the Guest Shell

The **guestshell disable** command shuts down and disables Guest Shell. When Guest Shell is disabled and the system is reloaded, Guest Shell remains disabled.

The **guestshell destroy** command removes the rootfs from the flash filesystem. All files, data, installed Linux applications and custom Python tools and utilities are deleted, and are not recoverable.

Accessing the Python Interpreter

Python can be used interactively or Python scripts can be run in the Guest Shell. Use the **guestshell run python** command to launch the Python interpreter in Guest Shell and open the Python terminal.

Note The **guestshell run** command is the IOS equivalent of running Linux executables, and when running a Python script from IOS, specify the absolute path. The following example shows how to specify the absolute path for the command:

```
Guestshell run python /flash/sample_script.py parameter1 parameter2
```

Configuration Examples for Guest Shell

Example: Managing the Guest Shell

The following example shows how to enable Guest Shell on a Catalyst 3850 Series Switch:

```
Device> enable
Device# guestshell enable
```

```
Management Interface will be selected if configured
Please wait for completion
Guestshell enabled successfully

Device# guestshell run python

Python 2.7.11 (default, Feb 21 2017, 03:39:40)
[GCC 5.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.

Device# guestshell run bash

[guestshell@guestshell ~]$

Device# guestshell disable

Guestshell disabled successfully

Device# guestshell destroy

Guestshell destroyed successfully
```

Sample VirtualPortGroup Configuration

When using the VirtualPortGroup interface for Guest Shell networking, the VirtualPortGroup interface must have a static IP address configured. The front port interface must be connected to the Internet and Network Address Translation (NAT) must be configured between the VirtualPortGroup and the front panel port.

The following is a sample VirtualPortGroup configuration:

```
Device> enable
Device# configure terminal
Device(config)# interface VirtualPortGroup 0
Device(config-if)# ip address 192.168.35.1 255.255.255.0
Device(config-if)# ip nat inside
Device(config-if)# no mop enabled
Device(config-if)# no mop sysid
Device(config-if)# exit
Device(config)# interface GigabitEthernet 0/0/3
Device(config-if)# ip address 10.0.12.19 255.255.0.0
Device(config-if)# ip nat outside
Device(config-if)# negotiation auto
Device(config-if)# exit
Device(config)# ip route 0.0.0.0 0.0.0.0 10.0.0.1
Device(config)# ip route 10.0.0.0 255.0.0.0 10.0.0.1
!Port forwarding to use ports for SSH and so on.
Device(config)# ip nat inside source static tcp 192.168.35.2 7023 10.0.12.19 7023 extendable
Device(config)# ip nat outside source list NAT_ACL interface GigabitEthernet 0/0/3 overload
Device(config)# ip access-list standard NAT_ACL
Device(config-std-nacl)# permit 192.168.0.0 0.0.255.255
Device(config-std-nacl)# exit
Device(config)# exit
Device#
```

Example: Guest Shell Usage

Example: Guest Shell Networking Configuration

For Guest Shell networking, the following configurations are required.

- Configure Domain Name System (DNS)
- Configure proxy settings
- Configure YUM or PIP to use proxy settings

Sample DNS Configuration for Guest Shell

The following is a sample DNS configuration for Guest Shell:

```
[guestshell@guestshell ~]$ cat/etc/resolv.conf
nameserver 192.0.2.1

Other Options:
[guestshell@guestshell ~]$ cat/etc/resolv.conf
domain cisco.com
search cisco.com
nameserver 192.0.2.1
search cisco.com
nameserver 198.51.100.1
nameserver 172.16.0.6
domain cisco.com
nameserver 192.0.2.1
nameserver 172.16.0.6
nameserver 192.168.255.254
```

Example: Configuring Proxy Environment Variables

If your network is behind a proxy, configure proxy variables in Linux. If required, add these variables to your environment.

The following example shows how to configure your proxy variables:

```
[guestshell@guestshell ~]$cat /bootflash/proxy_vars.sh
export http_proxy=http://proxy.example.com:80/
export https_proxy=http://proxy.example.com:80/
export ftp_proxy=http://proxy.example.com:80/
export no_proxy=example.com
export HTTP_PROXY=http://proxy.example.com:80/
export HTTPS_PROXY=http://proxy.example.com:80/
export FTP_PROXY=http://proxy.example.com:80/
guestshell ~] source /bootflash/proxy_vars.sh
```

Example: Configuring Yum and PIP for Proxy Settings

The following example shows how to use Yum for setting proxy environment variables:

```
cat /etc/yum.conf | grep proxy
[guestshell@guestshell~]$ cat/bootflash/yum.conf | grep proxy
proxy=http://proxy.example.com:80/
```

PIP install picks up environment variable used for proxy settings. Use sudo with -E option for PIP installation. If the environment variables are not set, define them explicitly in PIP commands as shown in following example:

```
sudo pip --proxy http://proxy.example.com:80/install requests
sudo pip install --trusted-bost pypi.example.com --index-url
http://pypi.example.com/simple requests
```

The following example shows how to use PIP install for Python:

```
Sudo -E pip install requests
[guestshell@guestshell ~]$ python
Python 2.17.11 (default, Feb 3 2017, 19:43:44)
[GCC 4.7.0] on linux2
Type "help", "copyright", "credits" or "license" for more information
>>>import requests
```

Additional References for Guest Shell

Related Documents

Related Topic	Document Title
Python module	• *CLI Python Module*
Zero-Touch Provisioning	

MIBs

MB	MIBs Link
	To locate and download MIBs for selected platforms, Cisco IOS releases, and feature sets, use Cisco MIB Locator found at the following URL: http://www.cisco.com/go/mibs

Technical Assistance

Description	Link
The Cisco Support website provides extensive online resources, including documentation and tools for troubleshooting and resolving technical issues with Cisco products and technologies. To receive security and technical information about your products, you can subscribe to various services, such as the Product Alert Tool (accessed from Field Notices), the Cisco Technical Services Newsletter, and Really Simple Syndication (RSS) Feeds. Access to most tools on the Cisco Support website requires a Cisco.com user ID and password.	http://www.cisco.com/support

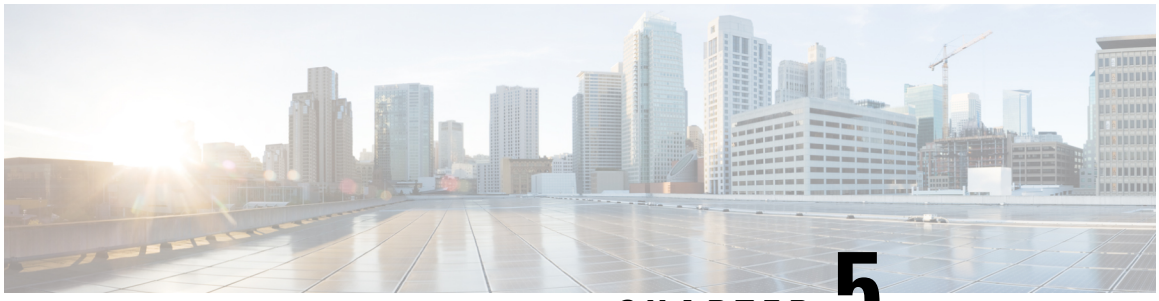
Feature Information for Guest Shell

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 8: Feature Information for Guest Shell

Feature Name	Release	Feature Information
Guest Shell		Guest Shell is a secure container that is an embedded Linux environment that allows customers to develop and run Linux and custom Python applications for automated control and management of Cisco switches. It also includes the automated provisioning (Day zero) of systems. This container shell provides a secure environment, decoupled from the host device, in which users can install scripts or software packages and run them. In Cisco IOS XE Everest 16.5.1a, this feature was implemented on the following platforms:



CHAPTER 5

Python API

Python programmabililty supports Python APIs.

- Finding Feature Information, on page 39
- Using Python, on page 39

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest caveats and feature information, see Bug Search Tool and the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the feature information table at the end of this module.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to https://cfnng.cisco.com/. An account on Cisco.com is not required.

Using Python

Cisco Python Module

Cisco provides a Python module that provides access to run EXEC and configuration commands. You can display the details of the Cisco Python module by entering the **help()** command. The **help()** command displays the properties of the Cisco CLI module.

The following example displays information about the Cisco Python module:

```
Device# guestshell run python

Python 2.7.5 (default, Jun 17 2014, 18:11:42)
[GCC 4.8.2 20140120 (Red Hat 4.8.2-16)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> >>> from cli import cli,clip,configure,configurep, execute, executep
>>> help(configure)
Help on function configure in module cli:

configure(configuration)
Apply a configuration (set of Cisco IOS CLI config-mode commands) to the device
and return a list of results.
```

```
configuration = '''interface gigabitEthernet 0/0
no shutdown'''

# push it through the Cisco IOS CLI.
try:
results = cli.configure(configuration)
print "Success!"
except CLIConfigurationError as e:
print "Failed configurations:"
for failure in e.failed:
print failure

Args:
configuration (str or iterable): Configuration commands, separated by newlines.

Returns:
list(ConfigResult): A list of results, one for each line.

Raises:
CLISyntaxError: If there is a syntax error in the configuration.

>>> help(configurep)
Help on function configurep in module cli:

configurep(configuration)
Apply a configuration (set of Cisco IOS CLI config-mode commands) to the device
and prints the result.

configuration = '''interface gigabitEthernet 0/0
no shutdown'''

# push it through the Cisco IOS CLI.
configurep(configuration)

Args:
configuration (str or iterable): Configuration commands, separated by newlines.
>>> help(execute)
Help on function execute in module cli:

execute(command)
Execute Cisco IOS CLI exec-mode command and return the result.

command_output = execute("show version")

Args:
command (str): The exec-mode command to run.

Returns:
str: The output of the command.

Raises:
CLISyntaxError: If there is a syntax error in the command.

>>> help(executep)
Help on function executep in module cli:

executep(command)
Execute Cisco IOS CLI exec-mode command and print the result.

executep("show version")

Args:
command (str): The exec-mode command to run.
```

```
>>> help(cli)
Help on function cli in module cli:

cli(command)
    Execute Cisco IOS CLI command(s) and return the result.

    A single command or a delimited batch of commands may be run. The
    delimiter is a space and a semicolon, " ;". Configuration commands must be
    in fully qualified form.

    output = cli("show version")
    output = cli("show version ; show ip interface brief")
    output = cli("configure terminal ; interface gigabitEthernet 0/0 ; no shutdown")

    Args:
        command (str): The exec or config CLI command(s) to be run.

    Returns:
        string: CLI output for show commands and an empty string for
            configuration commands.

    Raises:
        errors.cli_syntax_error: if the command is not valid.
        errors.cli_exec_error: if the execution of command is not successful.


>>> help(clip)
Help on function clip in module cli:

clip(command)
    Execute Cisco IOS CLI command(s) and print the result.

    A single command or a delimited batch of commands may be run. The
    delimiter is a space and a semicolon, " ;". Configuration commands must be
    in fully qualified form.

    clip("show version")
    clip("show version ; show ip interface brief")
    clip("configure terminal ; interface gigabitEthernet 0/0 ; no shutdown")

    Args:
        command (str): The exec or config CLI command(s) to be run.
```

Cisco Python Module to Execute IOS CLI Commands

Note Guest Shell must be enabled for Python to run. For more information, see the *Guest Shell* chapter.

The Python programming language uses six functions that can execute CLI commands. These functions are available from the Python CLI module. To use these functions, execute the **import cli** command. The **ip http server** command must be enabled for these functions to work.

Arguments for these functions are strings of CLI commands. To execute a CLI command through the Python interpreter, enter the CLI command as an argument string of one of the following six functions:

- **cli.cli(command)**—This function takes an IOS command as an argument, runs the command through the IOS parser, and returns the resulting text. If this command is malformed, a Python exception is raised. The following is sample output from the **cli.cli(command)** function:

```
>>> import cli
>>> cli.clip('configure terminal; interface loopback 10; ip address
10.10.10.10 255.255.255.255')
*Mar 13 18:39:48.518: %LINEPROTO-5-UPDOWN: Line protocol on Interface Loopback10, changed
 state to up
>>> cli.clip('show clock')
'\n*18:11:53.989 UTC Mon Mar 13 2017\n'
>>> output=cli.cli('show clock')
>>> print(output)
*18:12:04.705 UTC Mon Mar 13 2017
```

- **cli.clip(command)**—This function works exactly the same as the **cli.cli(command)** function, except that it prints the resulting text to *stdout* rather than returning it. The following is sample output from the **cli.clip(command)** function:

```
>>> cli
>>> cli.clip('configure terminal; interface loopback 11; ip address
10.11.11.11 255.255.255.255')
*Mar 13 18:42:35.954: %LINEPROTO-5-UPDOWN: Line protocol on Interface Loopback11, changed
 state to up
*Mar 13 18:42:35.954: %LINK-3-UPDOWN: Interface Loopback11, changed state to up
>>> cli.clip('show clock')
*18:13:35.313 UTC Mon Mar 13 2017
>>> output=cli.clip('show clock')
*18:19:26.824 UTC Mon Mar 13 2017
>>> print (output)
None
```

- **cli.execute(command)**—This function executes a single EXEC command and returns the output; however, does not print the resulting text No semicolons or newlines are allowed as part of this command. Use a Python list with a for-loop to execute this function more than once. The following is sample output from the **cli.execute(command)**

  function:

```
>>> cli.execute("show clock")
'15:11:20.816 UTC Thu Jun 8 2017'
>>>
>>> cli.execute('show clock'; 'show ip interface brief')
  File "<stdin>", line 1
    cli.execute('show clock'; 'show ip interface brief')
                          ^
SyntaxError: invalid syntax
>>>
```

- **cli.executep(command)**—This function executes a single command and prints the resulting text to *stdout* rather than returning it. The following is sample output from the **cli.executep(command)** function:

```
>>> cli.executep('show clock')
```

```
*18:46:28.796 UTC Mon Mar 13 2017
>>> output=cli.executep('show clock')
*18:46:36.399 UTC Mon Mar 13 2017
>>> print(output)
None
```

- **cli.configure(command)**—This function configures the device with the configuration available in commands. It returns a list of named tuples that contains the command and its result as shown below:

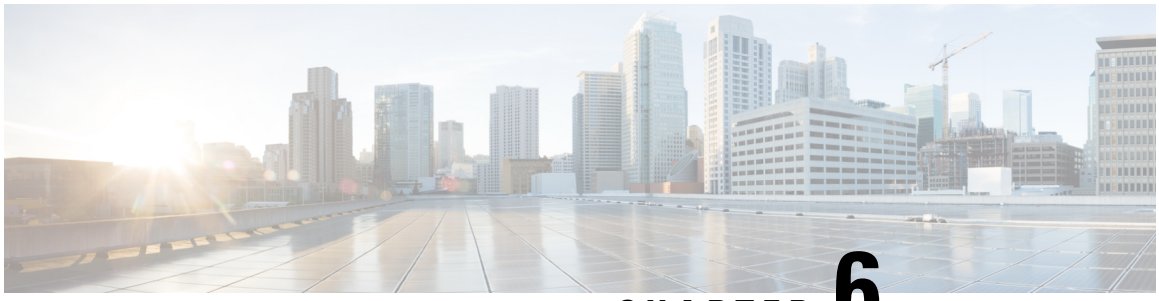
  ```
  [Think: result = (bool(success), original_command, error_information)]
  ```

  The command parameters can be in multiple lines and in the same format that is displayed in the output of the **show running-config** command. The following is sample output from the **cli.configure(command)** function:

  ```
  >>>cli.configure(["interface GigabitEthernet1/0/7", "no shutdown",
  "end"])
  [ConfigResult(success=True, command='interface GigabitEthernet1/0/7',
  line=1, output='', notes=None), ConfigResult(success=True, command='no shutdown',
  line=2, output='', notes=None), ConfigResult(success=True, command='end',
  line=3, output='', notes=None)]
  ```

- **cli.configurep(command)**—This function works exactly the same as the **cli.configure(command)** function, except that it prints the resulting text to *stdout* rather than returning it. The following is sample output from the **cli.configurep(command)** function:

  ```
  >>> cli.configurep(["interface GigabitEthernet1/0/7", "no shutdown",
  "end"])
  Line 1 SUCCESS: interface GigabitEthernet1/0/7
  Line 2 SUCCESS: no shut
  Line 3 SUCCESS: end
  ```



CHAPTER 6

CLI Python Module

Python Programmability provides a Python module that allows users to interact with IOS using CLIs.

- Finding Feature Information, on page 45
- Information About CLI Python Module, on page 45
- Updating the Cisco CLI Python Module, on page 48
- Additional References for the CLI Python Module, on page 48
- Feature Information for the CLI Python Module, on page 49

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest caveats and feature information, see Bug Search Tool and the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the feature information table at the end of this module.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to https://cfnng.cisco.com/. An account on Cisco.com is not required.

Information About CLI Python Module

About Python

The Cisco IOS XE devices support Python Version 2.7 in both interactive and non-interactive (script) modes within the Guest Shell. The Python scripting capability gives programmatic access to a device's CLI to perform various tasks and Zero Touch Provisioning or Embedded Event Manager (EEM) actions.

Python Scripts Overview

Python run in a virtualized Linux-based environment, Guest Shell. For more information, see the *Guest Shell* chapter. Cisco provides a Python module that allows user's Python scripts to run IOS CLI commands on the host device.

Interactive Python Prompt

When you execute the **guestshell run python** command on a device, the interactive Python prompt is opened inside the Guest Shell. The Python interactive mode allows users to execute Python functions from the Cisco Python CLI module to configure the device.

The following example shows how to enable the interactive Python prompt:

```
Device# guestshell run python

Python 2.7.5 (default, Jun 17 2014, 18:11:42)
[GCC 4.8.2 20140120 (Red Hat 4.8.2-16)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>

Device#
```

Python Script

Python scripts can run in non-interactive mode by providing the Python script name as an argument in the Python command. Python scripts must be accessible from within the Guest Shell. To access Python scripts from the Guest Shell, save the scripts in bootflash/flash that is mounted within the Guest Shell.

The following sample Python script uses different CLI functions to configure and print **show** commands:

```
Device# more flash:sample_script.py


import sys
import cli

intf= sys.argv[1:]
intf = ''.join(intf[0])

print "\n\n *** Configuring interface %s with 'configurep' function  *** \n\n" %intf
cli.configurep(["interface loopback55","ip address 10.55.55.55 255.255.255.0","no
shut","end"])

print "\n\n *** Configuring interface %s with 'configure' function  *** \n\n"
cmd='interface %s,logging event link-status ,end' % intf
cli.configure(cmd.split(','))

print "\n\n *** Printing show cmd with 'executep' function  *** \n\n"
cli.executep('show ip interface brief')

print "\n\n *** Printing show cmd with 'execute' function  *** \n\n"
output= cli.execute('show run interface %s' %intf)
print (output)

print "\n\n *** Configuring interface %s with 'cli' function  *** \n\n"
cli.cli('config terminal; interface %s; spanning-tree portfast edge default' %intf)

print "\n\n *** Printing show cmd with 'clip' function  *** \n\n"
cli.clip('show run interface %s' %intf)


To run a Python script from the Guest Shell, execute the guestshell run python
/flash/script.py command
at the device prompt.
The following example shows how to run a Python script from the Guest Shell:
```

The following example shows how to run a Python script from the Guest Shell:

```
Device# guestshell run python /flash/sample_script.py loop55

 *** Configuring interface loop55 with 'configurep' function  ***

Line 1 SUCCESS: interface loopback55
Line 2 SUCCESS: ip address 10.55.55.55 255.255.255.0
Line 3 SUCCESS: no shut
Line 4 SUCCESS: end

 *** Configuring interface %s with 'configure' function  ***


 *** Printing show cmd with 'executep' function  ***

Interface              IP-Address      OK? Method Status                Protocol
Vlan1                  unassigned      YES NVRAM  administratively down down
GigabitEthernet0/0     192.0.2.1       YES NVRAM  up                    up
GigabitEthernet1/0/1   unassigned      YES unset  down                  down
GigabitEthernet1/0/2   unassigned      YES unset  down                  down
GigabitEthernet1/0/3   unassigned      YES unset  down                  down
    :
         :
         :
Te1/1/4                unassigned      YES unset  down                  down
Loopback55             10.55.55.55     YES TFTP   up                    up
Loopback66             unassigned      YES manual up                    up


 *** Printing show cmd with 'execute' function  ***

Building configuration...
Current configuration : 93 bytes
!
interface Loopback55
 ip address 10.55.55.55 255.255.255.0
 logging event link-status
end

 *** Configuring interface %s with 'cli' function  ***

 *** Printing show cmd with 'clip' function  ***

Building configuration...
Current configuration : 93 bytes
!
interface Loopback55
 ip address 10.55.55.55 255.255.255.0
 logging event link-status
end
```

Supported Python Versions

Guest Shell is pre-installed with Python Version 2.7. Guest Shell is a virtualized Linux-based environment, designed to run custom Linux applications, including Python applications for automated control and

management of Cisco devices. Platforms with Montavista CGE7 support Python Version 2.7.11, and platforms with CentOS 7 support Python Version 2.7.5.

The following table provides information about Python versions and the supported platforms:

Table 9: Python Version Support

Python Version	Platform

Platforms with CentOS 7 support the installation of Redhat Package Manager (RPM) from the open source repository.

Updating the Cisco CLI Python Module

The Cisco CLI Python module and EEM module are pre-installed on devices. However, when you update the Python version by using either Yum or prepackaged binaries, the Cisco-provided CLI module must also be updated.

Note When you update to Python Version 3 on a device that already has Python Version 2, both versions of Python exist on the device. Use one of the following IOS commands to run Python:

- The **guestshell run python2** command enables Python Version 2.
- The **guestshell run python3** command enables Python Version 3.
- The **guestshell run python** command enables Python Version 2.

Use one of the following methods to update the Python version:

- Standalone tarball installation
- PIP install for the CLI module

Additional References for the CLI Python Module

Related Documents

Related Topic	Document Title
Guest Shell	
EEM Python Module	

Technical Assistance

Description	Link
The Cisco Support website provides extensive online resources, including documentation and tools for troubleshooting and resolving technical issues with Cisco products and technologies. To receive security and technical information about your products, you can subscribe to various services, such as the Product Alert Tool (accessed from Field Notices), the Cisco Technical Services Newsletter, and Really Simple Syndication (RSS) Feeds. Access to most tools on the Cisco Support website requires a Cisco.com user ID and password.	http://www.cisco.com/support

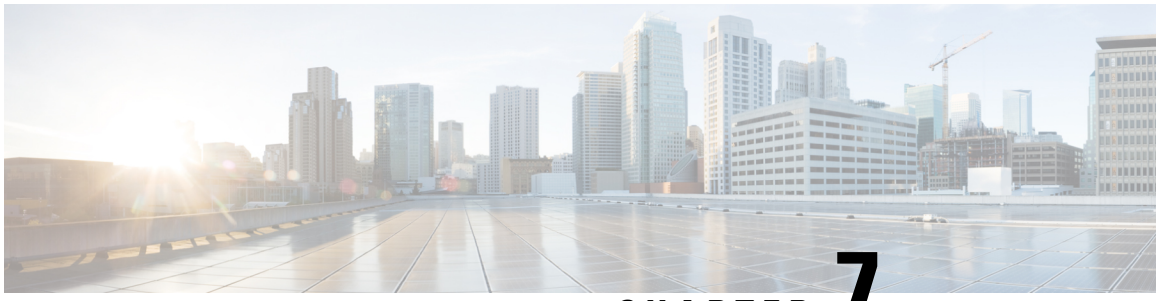
Feature Information for the CLI Python Module

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 10: Feature Information for the CLI Python Module

Feature Name	Release	Feature Information
CLI Python Module		Python programmabilty provides a Python module that allows users to interact with IOS using CLIs.



CHAPTER 7

EEM Python Module

Embedded Event Manager (EEM) policies support Python scripts. Python scripts can be executed as part of EEM actions in EEM applets.

- Finding Feature Information, on page 51
- Prerequisites for the EEM Python Module, on page 51
- Information About the EEM Python Module, on page 51
- How to Configure the EEM Python Policy, on page 54
- Additional References EEM Python Module, on page 59
- Feature Information for EEM Python Module, on page 60

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest caveats and feature information, see Bug Search Tool and the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the feature information table at the end of this module.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to https://cfnng.cisco.com/. An account on Cisco.com is not required.

Prerequisites for the EEM Python Module

Guest Shell must be working within the container. Guest Shell is not enabled by default. For more information see the *Guest Shell* feature.

Information About the EEM Python Module

Python Scripting in EEM

Embedded Event Manager (EEM) policies support Python scripts. You can register Python scripts as EEM policies, and execute the registered Python scripts when a corresponding event occurs. The EEM Python script has the same event specification syntax as the EEM TCL policy.

Configured EEM policies run within the Guest Shell. Guest Shell is a virtualized Linux-based environment, designed to run custom Linux applications, including Python for automated control and management of Cisco devices. The Guest Shell container provides a Python interpreter.

EEM Python Package

The EEM Python package can be imported to Python scripts for running EEM-specific extensions.

Note The EEM Python package is available only within the EEM Python script (The package can be registered with EEM, and has the EEM event specification in the first line of the script.) and not in the standard Python script (which is run using the Python script name).

The Python package includes the following application programming interfaces (APIs):

- Action APIs—Perform EEM actions and have default parameters.
- CLI-execution APIs—Run IOS commands, and return the output. The following are the list of CLI-execution APIs:
  - eem_cli_open()
  - eem_cli_exec()
  - eem_cli_read()
  - eem_cli_read_line()
  - eem_cli_run()
  - eem_cli_run_interactive()
  - eem_cli_read_pattern()
  - eem_cli_write()
  - eem_cli_close()
- Environment variables-accessing APIs—Get the list of built-in or user-defined variables. The following are the environment variables-accessing APIs:
  - eem_event_reqinfo ()-Returns the built-in variables list.
  - eem_user_variables()-Returns the current value of an argument.

Python-Supported EEM Actions

The Python package (is available only within the EEM script, and not available for the standard Python script) supports the following EEM actions:

- Syslog message printing
- Send SNMP traps
- Reload the box

- Switchover to the standby device
- Run a policy
- Track Object read
- Track Object Set
- Cisco Networking Services event generation

The EEM Python package exposes the interfaces for executing EEM actions. You can use the Python script to call these actions, and they are forwarded from the Python package via Cisco Plug N Play (PnP) to the action handler.

EEM Variables

An EEM policy can have the following types of variables:

- Event-specific built-in variables—A set of predefinied variables that are populated with details about the event that triggered the policy. The eem_event_reqinfo () API returns the builtin variables list. These variables can be stored in the local machine and used as local variables. Changes to local variables do not reflect in builtin variables.
- User-defined variables—Variables that can be defined and used in policies. The value of these variables can be referred in the Python script. While executing the script, ensure that the latest value of the variable is available. The eem_user_variables() API returns the current value of the argument that is provided in the API.

EEM CLI Library Command Extensions

The following CLI library commands are available within EEM for the Python script to work:

- eem_cli_close()—Closes the EXEC process and releases the VTY and the specified channel handler connected to the command.
- eem_cli_exec—Writes the command to the specified channel handler to execute the command. Then reads the output of the command from the channel and returns the output.
- eem_cli_open—Allocates a VTY, creates an EXEC CLI session, and connects the VTY to a channel handler. Returns an array including the channel handler.
- eem_cli_read()—Reads the command output from the specified CLI channel handler until the pattern of the device prompt occurs in the contents read. Returns all the contents read up to the match.
- eem_cli_read_line()—Reads one line of the command output from the specified CLI channel handler. Returns the line read.
- eem_cli_read_pattern()—Reads the command output from the specified CLI channel handler until the pattern that is to be matched occurs in the contents read. Returns all the contents read up to the match.
- eem_cli_run()—Iterates over the items in the *clist* and assumes that each one is a command to be executed in the enable mode. On success, returns the output of all executed commands and on failure, returns error.

- eem_cli_run_interactive()—Provides a sublist to the *clist* which has three items. On success, returns the output of all executed commands and on failure, returns the error. Also uses arrays when possible as a way of making things easier to read later by keeping expect and reply separated.
- eem_cli_write()—Writes the command that is to be executed to the specified CLI channel handler. The CLI channel handler executes the command.

How to Configure the EEM Python Policy

For the Python script to work, you must enable the Guest Shell. For more information, see the *Guest Shell* chapter.

Registering a Python Policy

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**event manager directory user policy** *path* **Example:** `Device(config)# event manager directory user policy flash:/user_library`	Specifies a directory to use for storing user library files or user-defined EEM policies. **Note** You must have a policy in the specified path. For example, in this step, the eem_script.py policy is available in the flash:/user_library folder or path.
Step 4	**event manager policy** *policy-filename* **Example:** `Device(config)# event manager policy eem_script.py`	Registers a policy with EEM. • The policy is parsed based on the file extension. If the file extension is .py, the policy is registered as Python policy. • EEM schedules and runs policies on the basis of an event specification that is contained within the policy itself. When the **event manager policy** command is invoked, EEM examines the policy and registers it to be run when the specified event occurs.

	Command or Action	Purpose
Step 5	**exit** **Example:** `Device(config)# exit`	Exits global configuration mode and returns to privileged EXEC mode.
Step 6	**show event manager policy registered** **Example:** `Device# show event manager policy registered`	Displays the registered EEM policies.
Step 7	**show event manager history events** **Example:** `Device# show event manager history events`	Displays EEM events that have been triggered.

Example

The following is sample output from the **show event manager policy registered** command:

```
Device# show event manager policy registered

No.  Class     Type    Event Type          Trap  Time Registered           Name
1    script    user    multiple            Off   Tue Aug 2 22:12:15 2016   multi_1.py
 1: syslog: pattern {COUNTER}
 2: none: policyname {multi_1.py} sync {yes}
 trigger delay 10.000
  correlate event 1 or event 2
  attribute tag 1 occurs 1
 nice 0 queue-priority normal maxrun 100.000 scheduler rp_primary Secu none


2    script    user    multiple            Off   Tue Aug 2 22:12:20 2016   multi_2.py
 1: syslog: pattern {COUNTER}
 2: none: policyname {multi_2.py} sync {yes}
 trigger
  correlate event 1 or event 2
 nice 0 queue-priority normal maxrun 100.000 scheduler rp_primary Secu none


3    script    user    multiple            Off   Tue Aug 2 22:13:31 2016   multi.tcl
 1: syslog: pattern {COUNTER}
 2: none: policyname {multi.tcl} sync {yes}
 trigger
  correlate event 1 or event 2
  attribute tag 1 occurs 1
 nice 0 queue-priority normal maxrun 100.000 scheduler rp_primary Secu none
```

Running Python Scripts as Part of EEM Applet Actions

Python Script: eem_script.py

An EEM applet can include a Python script with an action command. In this example, an user is trying to run a standard Python script as part of the EEM action, however; EEM Python package is not available in the standard Python script. The standard Python script in IOS has a package named *from cli import cli,clip* and this package can be used to execute IOS commands.

```
import sys
from cli import cli,clip,execute,executep,configure,configurep

intf= sys.argv[1:]
intf = ''.join(intf[0])


print ('This script is going to unshut interface %s and then print show ip interface
brief'%intf)

if intf == 'loopback55':
configurep(["interface loopback55","no shutdown","end"])
else :
cmd='int %s,no shut ,end' % intf
configurep(cmd.split(','))

executep('show ip interface brief')
```

This following is sample output from the **guestshell run python** command.

```
Device#  guestshell run python /flash/eem_script.py loop55

This script is going to unshut interface loop55 and then print show ip interface brief
Line 1 SUCCESS: int loop55
Line 2 SUCCESS: no shut
Line 3 SUCCESS: end
Interface IP-Address OK? Method Status Protocol
Vlan1 unassigned YES NVRAM administratively down down
GigabitEthernet0/0 5.30.15.37 YES NVRAM up up
GigabitEthernet1/0/1 unassigned YES unset down down
GigabitEthernet1/0/2 unassigned YES unset down down
GigabitEthernet1/0/3 unassigned YES unset down down
GigabitEthernet1/0/4 unassigned YES unset up up
GigabitEthernet1/0/5 unassigned YES unset down down
GigabitEthernet1/0/6 unassigned YES unset down down
GigabitEthernet1/0/7 unassigned YES unset down down
GigabitEthernet1/0/8 unassigned YES unset down down
GigabitEthernet1/0/9 unassigned YES unset down down
GigabitEthernet1/0/10 unassigned YES unset down down
GigabitEthernet1/0/11 unassigned YES unset down down
GigabitEthernet1/0/12 unassigned YES unset down down
GigabitEthernet1/0/13 unassigned YES unset down down
GigabitEthernet1/0/14 unassigned YES unset down down
GigabitEthernet1/0/15 unassigned YES unset down down
GigabitEthernet1/0/16 unassigned YES unset down down
GigabitEthernet1/0/17 unassigned YES unset down down
GigabitEthernet1/0/18 unassigned YES unset down down
GigabitEthernet1/0/19 unassigned YES unset down down
GigabitEthernet1/0/20 unassigned YES unset down down
GigabitEthernet1/0/21 unassigned YES unset down down
GigabitEthernet1/0/22 unassigned YES unset down down
```

```
GigabitEthernet1/0/23 unassigned YES unset up up
GigabitEthernet1/0/24 unassigned YES unset down down
GigabitEthernet1/1/1 unassigned YES unset down down
GigabitEthernet1/1/2 unassigned YES unset down down
GigabitEthernet1/1/3 unassigned YES unset down down
GigabitEthernet1/1/4 unassigned YES unset down down
Te1/1/1 unassigned YES unset down down
Te1/1/2 unassigned YES unset down down
Te1/1/3 unassigned YES unset down down
Te1/1/4 unassigned YES unset down down
Loopback55 10.55.55.55 YES manual up up

Device#
Jun 7 12:51:20.549: %LINEPROTO-5-UPDOWN: Line protocol on Interface Loopback55,
changed state to up
Jun 7 12:51:20.549: %LINK-3-UPDOWN: Interface Loopback55, changed state to up
```

The following is a sample script for printing messages to the syslog. This script must be stored in a file, copied to the file system on the device, and registered using the event manager policy file.

```
::cisco::eem::event_register_syslog tag "1"    pattern     COUNTER    maxrun 200

import eem
import time

eem.action_syslog("SAMPLE SYSLOG MESSAGE","6","TEST")
```

The following is sample script to print EEM environment variables. This script must be stored in a file, copied to the file system on the device, and registered using the event manager policy file.

```
::cisco::eem::event_register_syslog tag "1"    pattern     COUNTER    maxrun 200

import eem
import time

c = eem.env_reqinfo()


print "EEM Environment Variables"
for k,v in c.iteritems():
      print "KEY : " + k + str(" ---> ") + v

print "Built in Variables"
for i,j in a.iteritems() :
      print "KEY : " + i + str(" ---> ") + j
```

Adding a Python Script in an EEM Applet

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.

	Command or Action	Purpose
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**event manager applet** *applet-name* **Example:** `Device(config)# event manager applet interface_Shutdown`	Registers an applet with the Embedded Event Manager (EEM) and enters applet configuration mode.
Step 4	**event** [**tag** *event-tag*] **syslog pattern** *regular-expression* **Example:** `Device(config-applet)# event syslog pattern "Interface Loopback55, changed state to administratively down"`	Specifies a regular expression to perform the syslog message pattern match.
Step 5	**action** *label* **cli command** *cli-string* **Example:** `Device(config-applet)# action 0.0 cli command "en"`	Specifies the IOS command to be executed when an EEM applet is triggered.
Step 6	**action** *label* **cli command** *cli-string* [**pattern** *pattern-string*] **Example:** `Device(config-applet)# action 1.0 cli command "guestshell run python3 /bootflash/eem_script.py loop55"`	Specifies the action to be specified with the **pattern** keyword. • Specify a regular expression pattern string that will match the next solicited prompt.
Step 7	**end** **Example:** `Device(config-applet)# end`	Exits applet configuration mode and returns to privileged EXEC mode.
Step 8	**show event manager policy active** **Example:** `Device# show event manager policy active`	Displays EEM policies that are executing.
Step 9	**show event manager history events** **Example:** `Device# show event manager history events`	Displays the EEM events that have been triggered.

What to do next

The following example shows how to trigger the Python script configured in the task:

```
Device(config)# interface loopback 55
Device(config-if)# shutdown
Device(config-if)# end
Device#
```

```
Mar 13 10:53:22.358 EDT: %SYS-5-CONFIG_I: Configured from console by console
Mar 13 10:53:24.156 EDT: %LINK-5-CHANGED:  Line protocol on Interface Loopback55, changed
state to down
Mar 13 10:53:27.319 EDT: %LINK-3-UPDOWN:  Interface Loopback55, changed state to
administratively down
Enter configuration commands, one per line.  End with CNTL/Z.
Mar 13 10:53:35.38 EDT:  %LINEPROTO-5-UPDOWN: Line protocol on Interface Loopback55, changed
 state to up
*Mar 13 10:53:35.39 EDT %LINK-3-UPDOWN: Interface Loopback55, changed state to up
+++ 10:54:33 edi37(default) exec +++
show ip interface br
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet0/0/0   unassigned      YES unset  down                  down
GigabitEthernet0/0/1   unassigned      YES unset  down                  down
GigabitEthernet0/0/2   10.1.1.31       YES DHCP   up                    up
GigabitEthernet0/0/3   unassigned      YES unset  down                  down
GigabitEthernet0       192.0.2.1       YES manual up                    up
Loopback55             198.51.100.1    YES manual up                    up
Loopback66             172.16.0.1      YES manual up                    up
Loopback77             192.168.0.1     YES manual up                    up
Loopback88             203.0.113.1     YES manual up                    up
```

Additional References EEM Python Module

Related Documents

Related Topic	Document Title
EEM configuration	Embedded Event Manager Configuration Guide
EEM commands	Embedded Event Manager Command Reference
Guest Shell configuration	

Technical Assistance

Description	Link
The Cisco Support website provides extensive online resources, including documentation and tools for troubleshooting and resolving technical issues with Cisco products and technologies. To receive security and technical information about your products, you can subscribe to various services, such as the Product Alert Tool (accessed from Field Notices), the Cisco Technical Services Newsletter, and Really Simple Syndication (RSS) Feeds. Access to most tools on the Cisco Support website requires a Cisco.com user ID and password.	http://www.cisco.com/support

Feature Information for EEM Python Module

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

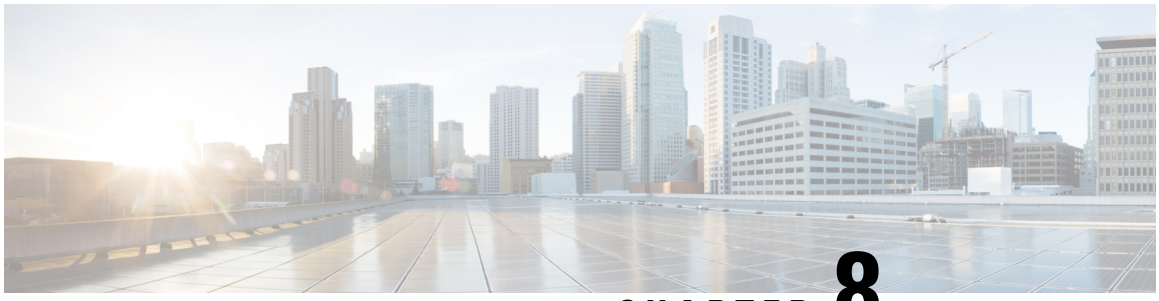
Table 11: Feature Information for EEM Python Module

Feature Name	Release	Feature Information
EEM Python Module	Cisco IOS XE Everest 16.5.1b	This feature supports Python scripts as EEM policies. No new commands were introduced.

PART III

Model-Driven Programmability

- Data Models, on page 63
- Operational Data Parser Polling, on page 69



CHAPTER 8

Data Models

- Finding Feature Information, on page 63
- Restrictions for Data Models , on page 63
- Information About Data Models, on page 63
- How to Configure Data Models, on page 64
- Additional References for Data Models, on page 67
- Feature Information for Data Models, on page 67

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest caveats and feature information, see Bug Search Tool and the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the feature information table at the end of this module.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to http://www.cisco.com/go/cfn. An account on Cisco.com is not required.

Restrictions for Data Models

The NETCONF feature is not supported on a device running dual IOSd configuration or software redundancy.

Information About Data Models

Introduction to Data Models - Programmatic and Standards-Based Configuration

The traditional way of managing network devices is by using Command Line Interfaces (CLIs) for configurational (configuration commands) and operational data (show commands). For network management, Simple Network Management Protocol (SNMP) is widely used, especially for exchanging management information between various network devices. Although CLIs and SNMP are heavily used, they have several

restrictions. CLIs are highly proprietary, and human intervention is required to understand and interpret their text-based specification. SNMP does not distinguish between configurational and operational data.

The solution lies in adopting a programmatic and standards-based way of writing configurations to any network device, replacing the process of manual configuration. Network devices running on Cisco IOS XE support the automation of configuration for multiple devices across the network using data models. Data models are developed in a standard, industry-defined language, that can define configuration and state information of a network.

Cisco IOS XE supports the Yet Another Next Generation (YANG) data modeling language. YANG can be used with the Network Configuration Protocol (NETCONF) to provide the desired solution of automated and programmable network operations. NETCONF (RFC 6241) is an XML-based protocol that client applications use to request information from and make configuration changes to the device. YANG is primarily used to model the configuration and state data used by NETCONF operations.

In Cisco IOS XE, model-based interfaces interoperate with existing device CLI, Syslog, and SNMP interfaces. These interfaces are optionally exposed northbound from network devices. YANG is used to model each protocol based on RFC 6020.

Note To access Cisco YANG models in a developer-friendly way, please clone the *GitHub repository*, and navigate to the *vendor/cisco* subdirectory. Models for various releases of IOS-XE, IOS-XR, and NX-OS platforms are available here.

NETCONF

NETCONF provides a simpler mechanism to install, manipulate, and delete the configuration of network devices.

It uses an Extensible Markup Language (XML)-based data encoding for the configuration data as well as the protocol messages.

NETCONF uses a simple RPC-based (Remote Procedure Call) mechanism to facilitate communication between a client and a server. The client can be a script or application typically running as part of a network manager. The server is typically a network device (switch or router). It uses Secure Shell (SSH) as the transport layer across network devices.

NETCONF also supports capability discovery and model downloads. Supported models are discovered using the *ietf-netconf-monitoring* model. Revision dates for each model are shown in the capabilities response. Data models are available for optional download from a device using the *get-schema* rpc. You can use these YANG models to understand or export the data model.

For more details, refer RFC 6241.

How to Configure Data Models

Configuring NETCONF

Before you begin

You must configure NETCONF-YANG as follows.

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**netconf-yang** **Example:** `Device (config)# netconf-yang`	Enables the NETCONF interface on your network device. **Note** After the initial enablement through the CLI, network devices can be managed subsequently through a model based interface. The complete activation of model-based interface processes may require up to 90 seconds.
Step 4	**exit** **Example:** `Device (config)# exit`	Exits global configuration mode.

Configuring NETCONF Options

Configuring SNMP

Enable the SNMP Server in IOS to enable NETCONF to access SNMP MIB data using YANG models generated from supported MIBs, and to enable supported SNMP traps in IOS to receive NETCONF notifications from the supported traps.

Perform the following steps:

Procedure

Step 1 Enable SNMP features in IOS.

Example:

```
configure terminal
logging history debugging
logging snmp-trap emergencies
logging snmp-trap alerts
logging snmp-trap critical
logging snmp-trap errors
logging snmp-trap warnings
logging snmp-trap notifications
```

```
logging snmp-trap informational
logging snmp-trap debugging
!
snmp-server community public RW
snmp-server trap link ietf
snmp-server enable traps snmp authentication linkdown linkup snmp-server enable traps syslog
snmp-server manager
exit
```

Step 2 After NETCONF-YANG starts, enable SNMP Trap support by sending the following RPC <edit-config> message to the NETCONF-YANG port.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <netconf-yang xmlns="http://cisco.com/yang/cisco-self-mgmt">
        <cisco-ia xmlns="http://cisco.com/yang/cisco-ia">
          <snmp-trap-control>
            <trap-list>
              <trap-oid>1.3.6.1.4.1.9.9.41.2.0.1</trap-oid>
            </trap-list>
            <trap-list>
              <trap-oid>1.3.6.1.6.3.1.1.5.3</trap-oid>
            </trap-list>
            <trap-list>
              <trap-oid>1.3.6.1.6.3.1.1.5.4</trap-oid>
            </trap-list>
          </snmp-trap-control>
        </cisco-ia>
      </netconf-yang>
    </config>
  </edit-config>
</rpc>
```

Step 3 Send the following RPC message to the NETCONF-YANG port to save the running configuration to the startup configuration.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="">
  <cisco-ia:save-config xmlns:cisco-ia="http://cisco.com/yang/cisco-ia"/>
</rpc>
```

Additional References for Data Models

Related Documents

Related Topic	Document Title
YANG data models for various release of IOS-XE, IOS-XR, and NX-OS platforms	To access Cisco YANG models in a developer-friendly way, please clone the *GitHub repository*, and navigate to the *vendor/cisco* subdirectory. Models for various releases of IOS-XE, IOS-XR, and NX-OS platforms are available here.

Standards and RFCs

Standard/RFC	Title
RFC 6020	*YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)*
RFC 6241	*Network Configuration Protocol (NETCONF)*

Technical Assistance

Description	Link
The Cisco Support website provides extensive online resources, including documentation and tools for troubleshooting and resolving technical issues with Cisco products and technologies. To receive security and technical information about your products, you can subscribe to various services, such as the Product Alert Tool (accessed from Field Notices), the Cisco Technical Services Newsletter, and Really Simple Syndication (RSS) Feeds. Access to most tools on the Cisco Support website requires a Cisco.com user ID and password.	http://www.cisco.com/support

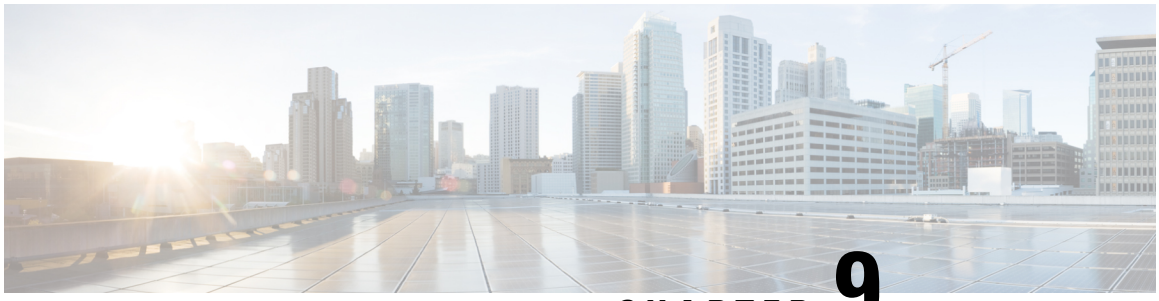
Feature Information for Data Models

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 12: Feature Information for Programmability: Data Models

Feature Name	Release	Feature Information
Data Models	Cisco IOS XE Denali 16.3.1	The Data Models feature facilitates a programmatic and standards-based way of writing configurations and reading operational data from network devices. The following command was introduced: **netconf-yang**.



CHAPTER 9

Operational Data Parser Polling

YANG data models enables you to read operational state data from devices.

- Finding Feature Information, on page 69
- Information About Operational Data, on page 69
- How to Enable Operational Data Parser Polling, on page 70
- Additional References for Operational Data Parser Polling, on page 72
- Feature Information for Operational Data Parser Polling, on page 73

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest caveats and feature information, see Bug Search Tool and the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the feature information table at the end of this module.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to https://cfnng.cisco.com/. An account on Cisco.com is not required.

Information About Operational Data

Operational Data Overview

You can use YANG data models to read operational state data from a device. The operational data allows you to determine the current state and behavior of a device, similar to IOS **show** commands.

You can perform NETCONF GET operations to retrieve read-only operational state data from a system. You must enable NETCONF, activate data parsers (where applicable), and then retrieve the data through an appropriate YANG model.

The *How to Configure Operational Data* section provides information on configuring operational data through a programmable interface and the CLI.

Operational Data Parsers and Corresponding YANG Models

There are two types of operational data parsers; one that is always on, and the other that must be configured to poll operational data at regular intervals. For the first type of operational data parser, no configuration is required. Data is always fetched from the device during a NETCONF GET request. These data parsers do not have a polling-interval, and operational data is updated as soon as a change occurs.

The second type of operational data parsers must be activated either via the CLI or a NETCONF message (For more information, see the *How to Enable Operational Data Parser Polling* section.). The operational data for these types of parsers is polled at regular polling intervals and this information is retrieved during a NETCONF GET request.

The following table lists the data parsers that must be activated, and the corresponding YANG model where the operational data is stored.

Table 13: Operational Data Parsers to be Activated and Corresponding Yang Models

Operational Data Parser Name	YANG Model to Access Operational Data
BGP	Cisco-IOS-XE-bgp-oper.yang
BFD	Cisco-IOS-XE-bfd-oper.yang
DiffServ	ietf-diffserv-target.yang
FlowMonitor	Cisco-IOS-XE-flow-monitor-oper.yang
IPRoute	ietf-routing.yang
MPLSLForwarding	Cisco-IOS-XE-mpls-fwd-oper.yang
MPLSLDPNeighbor	Cisco-IOS-XE-mpls-ldp.yang
MPLSStaticBinding	common-mpls-static.yang
OSPF	ietf-ospf.yang
PlatformSoftware	Cisco-IOS-XE-platform-software-oper.yang

How to Enable Operational Data Parser Polling

Enabling Operational Data Parser Polling Through a Programmable Interface

Perform this task to enable operational data parser polling through a programmable interface:

1. After enabling NETCONF-YANG, send an <edit-config> remote procedure call (RPC) using cisco-odm.yang (available in the GitHub Repository) to enable operational data polling. When the polling is enabled, all operational data parsers are activated by default. The default polling-interval of each parser is 120 seconds (120000 milliseconds). The polling interval decides the frequency at which the parser obtains the operational data and updates the corresponding YANG model in the datastore.

2. After operational data polling is enabled, send a <get> RPC to obtain the operational data. Use the parser-to-YANG model mapping to determine which operational YANG model should be used to retrieve the operational data. The following RPC reply fetches access control list (ACL) operational data using Cisco-IOS-XE-acl-oper.yang:

```
     CORRESPONDING RPC REPLY:
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
    <data>
        <access-lists xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-acl-oper">
            <access-list>
                <access-control-list-name>TEST</access-control-list-name>
                <access-list-entries>
                    <access-list-entry>
                        <rule-name>10</rule-name>
                        <access-list-entries-oper-data>
                        <match-counter>100</match-counter>
                    </access-list-entry>
                    <access-list-entry>
                        <rule-name>20</rule-name>
                        <access-list-entries-oper-data>
                        <match-counter>122</match-counter>
                    </access-list-entry>
                </access-list-entries>
            </access-list>
        </access-lists>
    </data>
</rpc-reply>
```

Note For more information, see the cisco-odm.yang model in the *GitHub repository*.

Enabling Operational Data Parser Polling Through the CLI

After enabling NETCONF-YANG, perform this task to enable operational data parser polling and to adjust the polling interval.

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**netconf-yang cisco-odm polling-enable** **Example:**	Enables operational data polling.

	Command or Action	Purpose
	`Device(config)# netconf-yang cisco-odm polling-enable`	
Step 4	**netconf-yang cisco-odm actions** *action-name* **Example:** `Device(config)# netconf-yang cisco-odm actions OSPF`	Enables the specified action, and enters ODM-action configuration mode. • Specify the operational data parser name to retrieve operational data.
Step 5	**mode poll** **Example:** `Device(config-odm-action)# mode poll`	Configures the data parser in poll mode.
Step 6	**polling-interval** *seconds* **Example:** `Device(config-odm-action)# polling-interval 1000`	Changes the default parser-polling interval. • To stop the parser from polling data, configure the **mode none** command.
Step 7	**end** **Example:** `Device(config-odm-action)# end`	Exits ODM-action configuration mode and returns to privileged EXEC mode.

What to do next

After enabling operational data polling, send a <get> RPC to obtain operational data from the device.

Additional References for Operational Data Parser Polling

Related Documents

Related Topic	Document Title
YANG data models for Cisco IOS XE	To access Cisco YANG models in a developer-friendly way, please clone the *GitHub repository*, and navigate to the *vendor/cisco* subdirectory.

MIBs

MB	MIBs Link
	To locate and download MIBs for selected platforms, Cisco IOS releases, and feature sets, use Cisco MIB Locator found at the following URL: http://www.cisco.com/go/mibs

Technical Assistance

Description	Link
The Cisco Support website provides extensive online resources, including documentation and tools for troubleshooting and resolving technical issues with Cisco products and technologies. To receive security and technical information about your products, you can subscribe to various services, such as the Product Alert Tool (accessed from Field Notices), the Cisco Technical Services Newsletter, and Really Simple Syndication (RSS) Feeds. Access to most tools on the Cisco Support website requires a Cisco.com user ID and password.	http://www.cisco.com/support

Feature Information for Operational Data Parser Polling

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 14: Feature Information for Operational Data Parser Polling

Feature Name	Release	Feature Information
Operational Data Parser Polling	Cisco IOS XE Everest 16.5.1a	YANG data models, enables you to read operational state data from a device. In Cisco IOS XE Everest 16.5.1a, this feature was implemented on the following platforms: